Introduction à Scilab

Scilab est un logiciel libre, disponible gratuitement sur le site `www.scilab.org` .

Ce premier thème, uniquement sous forme de séances pratiques à faire par binôme, consiste à prendre en main le logiciel Scilab avec les principales fonctionnalités qui seront utilisées tout au long du semestre.

Suivez le déroulement de cette fiche, testez les différents exemples présentés.
Différents exercices sont proposés au cours de cette séance, entraînez-vous à les faire.

1 - Démarrage de Scilab

Au début de chaque séance de TP, il vous faut créer un répertoire où vous stockerez les différents fichiers utilisés lors de la séance.

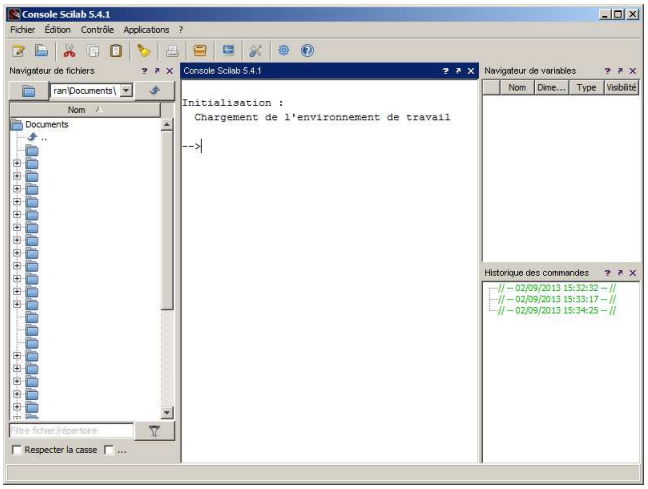
Pour cette première séance, ouvrez le `Poste de travail` puis le répertoire nommé `Z:` qui correspond à votre répertoire personnel sur lequel vous pouvez mettre vos propres documents et que vous retrouverez tel quel lors de chaque séance sur machine.

Dans ce répertoire personnel, créez un nouveau répertoire nommé `MAP101` puis dans celui-ci un nouveau répertoire nommé `TP1`.
Au début de chaque séance suivante, il suffira de créer un nouveau répertoire à coté du répertoire `TP1`.

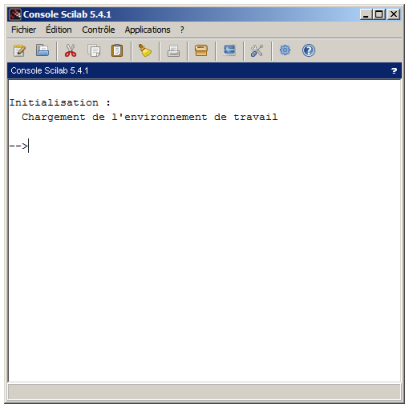
Démarrez SCILAB : double-cliquez sur l'icône correspondant ou utilisez le menu `Programmes` de Windows.

La première fois, la fenêtre Scilab doit s'afficher avec une *disposition intégrée* (voir la figure de gauche ci-dessous). Cette disposition permet d'accèder à de nombreuses options de Scilab.

Il est aussi possible de travailler en disposition simple où les différents outils de Scilab apparaissent dans des fenêtres séparées.



Disposition intégrée



Disposition simple

La fenêtre principale (appelée par la suite *console*) s'ouvre avec différents menus et icônes correspondants et la partie *console* où on peut entrer au clavier différentes commandes et où l'affichage de certains résultats est fait.

La première chose à faire est de vous placer dans le répertoire que vous avez créé précédemment afin de stocker les différents fichiers de la séance : dans le menu *Fichier*, sélectionnez l'item *Changer le répertoire courant ...* puis dans le dialogue, placez-vous dans le répertoire voulu et cliquez sur le bouton `Ok` .
On peut à tout moment changer de répertoire courant.

2 - Travailler avec Scilab

2.1 - Mode interactif

On peut utiliser SCILAB comme une calculette interactive, en mode console.
Exemple : Dans la console, tapez les instructions suivantes :

```
2.5*4-7/2
exp(2)
%e^2
3^2 , cos(%pi/3)
```

➪ les fonctions mathématiques usuelles sont définies
➪ différentes constantes sont définies, on utilisera notamment. la constante $\pi \simeq 3,1416$ notée `%pi` et la constante e $\simeq 2,7182$ notée `%e`
➪ notez le role de la virgule (`,`)

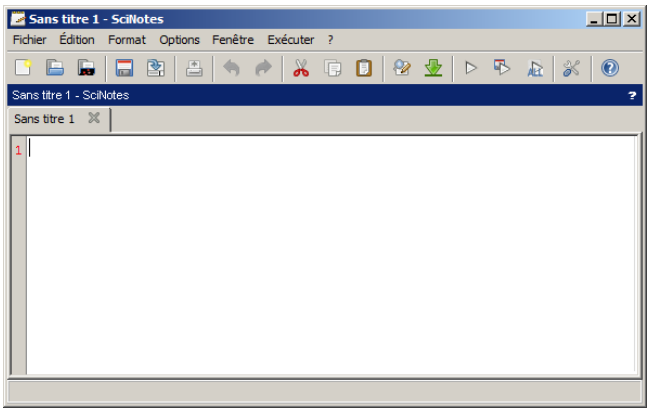
Il est possible d'utiliser des variables afin de stocker le résultat d'un calcul afin de pouvoir s'en servir ensuite.
Exemple : Dans la console, tapez les instructions suivantes :

```
H = 2; R = 1.5; V = %pi * R*H
H, R, V
```

➪ notez le rôle du point-virgule (`;`) par rapport à la virgule (`,`).
➪ on peut éventuellement réafficher, modifier et réexécuter les différentes instructions entrées précédemment en utilisant les touches *flèches haut* ↑ et *bas* ↓ du clavier.

2.2 - Utilisation de fichiers d'instructions

L'utilisation de la console peut devenir fastidieuse dès que le nombre d'instructions à écrire augmente. Il est alors préférable d'écrire les instructions dans un éditeur, pour ensuite les exécuter, les modifier et éventuellement les sauvegarder dans des fichiers qui pourront être réutilisés plus tard.
Pour utiliser l'éditeur, cliquez sur l'icône 
ou bien sélectionnez l'item *SciNotes* du menu *Application* :
l'éditeur de texte intégré de Scilab s'ouvre.



Dans la fenêtre de l'éditeur, tapez les instructions suivantes :

```
a = 7
b = 2
c = a/b
```

et sauvegardez-le (menu *Fichier*, item *Sauvegarder sous ...*) en le nommant `prog1.sce`.
⇨ par convention, les scripts Scilab contenant une suite d'instructions ont un nom avec le suffixe `.sce`

Il est alors possible d'exécuter une suite d'instructions en les sélectionnant à la souris dans l'éditeur puis en tapant simultanément les touches `Ctrl` et `E`,
ou bien d'exécuter l'ensemble du fichier en tapant simultanément les touches `Ctrl`, `Majuscule` et `E`.
Exemple : Testez ces deux possibilités avec le fichier `prog1.sce`

⇨ remarquez que dans le cas de l'exécution de l'ensemble du fichier, rien n'est écrit à l'écran. Dans ce cas, la présence ou l'absence de points-virgules ne changent rien, il faut alors explicitement utiliser la procédure `disp` afin d'afficher une valeur.
Exemple : Complétez le script `prog1.sce` en rajoutant l'instruction

```
disp(c)
```

à la fin du fichier, puis refaites une exécution complète : la valeur de `c` est écrite dans la console.

Une fois créé et sauvegardé, un fichier-script Scilab peut être réutilisé par la suite.
Exemple : Fermez l'éditeur de texte, puis dans la console, tapez l'instruction

```
exec("prog1.sce")
```

et ensuite, tapez l'instruction

```
exec("prog1.sce", -1)
```

et notez la différence entre les deux.
⇨ on peut aussi utiliser l'item *Exécuter ...* du menu *Fichier* de la fenêtre *console*.

2.3 - Aide en ligne

Pour avoir de l'aide sur les différentes fonctionnalités et instructions de Scilab, utilisez l'instruction `help` ou cliquez sur l'icône  .

Pour avoir l'aide sur une instruction particulière, utilisez l'onglet *loupe* du dialoque d'aide ou tapez l'instruction `help` *instruction*.

Exemple : Pour avoir l'aide sur l'instruction `clear`, tapez l'instruction `help clear`

3 - Valeurs et tableaux

3.1 - Nombres, booléens et chaînes de caractères

Scilab permet de manipuler des valeurs numériques, des booléens et des chaines de caractères.

• **Valeur numérique** : les valeurs numériques permettent de manipuler des réels (et donc aussi des entiers). On peut utiliser avec des valeurs numériques les opérateurs classiques (`+` *addition*, `-` *soustraction* ou *opposé*, `*` *multiplication*, `/` *division*, `^` *puissance*).

Par exemple, testez les instructions suivantes

```
a = 3
a+4, 7-a, -a, a*5, 1/a, a^2
```

• **Booléen** : une valeur booléenne est une valeur qui peut être soit *vrai*, soit *faux*. Les valeurs

booléennes servent entre autres dans des programmes à faire des tests, et exécuter ou non une suite d'instructions suivant la valeur booléenne d'une expression conditionnelle en comparant par exemple deux valeurs numériques.
Testez les instructions suivantes et observez les résultats obtenus :

```
a = 3, b = 3
a==b  // a EGAL A b
a~=b  // a DIFFERENT DE b
a<b   // a INFERIEUR STRICTEMENT A b
a<=b  // a INFERIEUR OU EGAL A b
a>b   // a SUPERIEUR STRICTEMENT A b
a>=b  // a SUPERIEUR OU EGAL A b
```

⇨ le résultat de chaque comparaison entre `a` et `b` est soit *VRAI* (affiché `T`), soit *FAUX* (affiché `F`).
⇨ sur une ligne d'instruction, ce qui suit les deux symboles `//` est un commentaire. On peut ensuite construire des expression booléennes plus complexes en utilisant les trois opérateurs booléens classiques (`&` *ET* , `|` *OU* , `~` *NON*).
Testez les instructions suivantes et observez les résultats obtenus :

```
a = 3, b = 3, c = 4
~(a==b)
(a<b) | (b<c)
(a<c) & (b+1<c)
```

• **Chaîne de caractères** Scilab permet aussi de manipuler des chaînes de caractères. Une chaîne de caractères est délimitée par des *double-quotes* (") . On se servira des chaines de caractères notamment pour la définition de fonctions, ainsi que pour certaines options graphiques.
Testez les instructions suivantes et observez les résultats obtenus :

```
s1 = "MAP", s2 = "101"
disp(s1+s1+s1)
disp(s1+s2)
```

⇨ pour deux chaînes de caractères, l'opérateur `+` permet de les *concaténer*.

3.2 - Définition, création de vecteurs et matrices

L'un des intérets du logiciel Scilab est de pouvoir effectuer des calculs sur des tableaux de valeurs. Scilab permet de manipuler des variables de type *tableau* puis d'effectuer des opérations sur celles-ci.

• **Vecteur** Un *vecteur* est un tableau formé d'une seule ligne ou d'une seule colonne.
Exemple : Avec l'éditeur de texte, créez un nouveau fichier nommé `tableaux.sce` avec les instructions suivantes :

```
v1 = [2 3 5 7 11 13]; // création d'un vecteur-ligne : séparer
v2 = [1,3,5];         // les valeurs par des espaces ou des virgules
w1 = [0;3;5;6];       // création d'un vecteur-colonne : séparer
                      // les valeurs par des points-virgules
disp(v1), disp(v2), disp(w1) // afficher les vecteurs
```

puis exécutez-les.

• **Matrice** Une *matrice* est un tableau avec un nombre quelconque de ligne et un nombre

quelconque de colonne (un vecteur est une matrice particulière).
Exemple : Complétez le fichier `tableaux.sce` en ajoutant les instructions suivantes :

```
A = [1 2;3 4;5 6;7 8]; // matrice avec 4 lignes et 2 colonnes
B = [4 6 -2;5 8 3];    // matrice avec 2 lignes et 3 colonnes
C = [1 2;0 3];         // matrice carrée avec 2 lignes et 2 colonnes
```

exécutez-les, puis dans la console Scilab, observez le résultat des instructions suivantes :
`[v1 v1]` `[v1;v1]` `[w1 w1 w1]` `[w1;w1;w1]` `[v2 v1]` `[v1 ; v2 v2]` `[w1 , [v2;v2;v2;v2]]`
`[C C C]` `[C;C;C]` `[A;C]`
puis testez les instructions suivantes :
`[v1 w1]` `[v2;v1]` `[w1;v2]` `[A B]` `[A;B]` `[A C]`
⇨ on peut assembler des tableaux en un tableau plus grand uniquement si les dimensions concordent.

3.3 - Accès aux éléments d'un tableau

Pour accèder à un élément d'un tableau, il suffit d'utiliser le ou les indices correspondants (chaque indice est un entier supérieur ou égal à 1) :

— pour un vecteur, chaque élement est repéré par un indice
Exemple : Observez les valeurs des instructions suivantes :

```
v1(1), v1(2) , v2(3) , w1(1) , w1(4)
```

— pour une matrice, chaque élement est repéré par deux indices, le premier est l'indice de ligne et le second est l'indice de colonne
Exemple : Observez le résultat des instructions suivantes :

```
A(1,1) , A(1,2) , A(2,1) , B(2,3) , C(1,2)
```

— on peut aussi extraire une ligne ou une colonne particulière d'une matrice.
Exemple : Observez le résultat des instructions suivantes :

```
A(1,:) , A(3,:) , A(:,2) , B(:,3) , C(1,:)
```

3.4 - Création de tableaux particuliers

Scilab fournit différentes procédures afin de créer des tableaux particuliers.

— l'instruction `zeros(m,n)` crée un tableau avec m lignes et n colonnes et dont toutes les valeurs sont égales à 0.
Exemple : Testez les instructions suivantes :

```
zeros(2,3) , zeros(3,5) , zeros(1,7) , zeros(3,1)
```

— l'instruction `ones(m,n)` crée un tableau avec m lignes et n colonnes et dont toutes les valeurs sont égales à 1.
Exemple : Testez les instructions suivantes :

```
ones(2,3) , ones(3,5) , ones(1,6) , ones(4,1)
```

— l'instruction `linspace(a,b,n)` crée un vecteur-ligne formé des n valeurs équiréparties entre `a` et `b`.
Exemple : Testez les instructions suivantes :

```
linspace(1,5,5) , linspace(0,4,9) , linspace(10,1,4)
```

⇨ on remarque que le *pas* (la différence entre deux valeurs consécutives) est donnée par la formule $\texttt{pas} = (\texttt{b} - \texttt{a})/(\texttt{n} - 1)$.

— si on souhaite construire un vecteur-ligne en spécifiant le *pas*, on utilisera alors la notation `a:pas:b`
Exemple : Testez les instructions suivantes :

```
1:1:5 , 0:0.5:4 , 10:-3:1 , 2:3:10 , 1:5 , 0:9 ,
```

⇨ si le *pas* n'est pas spécifié (`a:b`), il est égal à 1.

Exercice 1 :

en utilisant les instructions vues précédemment, écrire (si possible de la manière la plus simple) les instructions pour créer les tableaux suivants :

$$v1 = \begin{pmatrix} 5 & 6 & 7 & 8 & 9 & 10 \end{pmatrix} \quad v2 = \begin{pmatrix} 0 & 0 & 0 & 5 & 6 & 7 & 8 & 9 & 10 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$v3 = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 9 & 7 & 5 & 3 & 1 \end{pmatrix}$$

$$M1 = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix} \quad M2 = \begin{pmatrix} 1 & 3 & 5 & 7 & 9 \\ 8 & 6 & 4 & 2 & 0 \\ 8 & 6 & 4 & 2 & 0 \end{pmatrix} \quad M3 = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 2 & 3 & 4 & 5 \end{pmatrix}$$

3.5 - Opérations arithmétiques sur les tableaux

Une fois un tableau défini, on peut réaliser certaines opérations arithmétiques sur l'ensemble des éléments d'un tableau ou entre tableaux de même dimensions.
Exemple : Testez les instructions suivantes :

```
M = [1 2 3;4 5 6]
M + 2 // ajoute 2 à tous les éléments de M
M - 3 // retranche 3 à tous les éléments de M
M * (-4) // multiplie tous les éléments de M par -4
M / 10 // divise tous les éléments de M par 10
```

On peut aussi effectuer des opérations arithmétiques terme à terme entre deux tableaux de même dimensions.
Exemple : Testez les instructions suivantes :

```
M = [1 2 3;4 5 6] , P = [4 7 1;0 2 8]
M + P , P + M , M - P , P - M
```

Par contre si on souhaite multiplier ou diviser terme à terme les éléments de deux tableaux, il faudra utiliser les opérateurs `.*` et `./` au lieu de `*` et `/`
Exemple : Testez les instructions suivantes :

```
M .* P , P .* M , P ./ M
1 ./ M   // les inverses des elements de M
```

⇨ il est conseillé de mettre un espace AVANT et APRÈS chaque opérateur.

De même, on peut appliquer la puissance terme à terme en utilisant l'opérateur `.^`
Exemple : Testez les instructions suivantes :

```
v1 = 0:9      // les 10 premiers entiers
v2 = v1 .^ 2 // les 10 premiers carrés
v3 = v1 .^ 3 // les 10 premiers cubes
w  = 2 .^ v1 // les 10 premieres puissances de 2
```

⇨ pour effectuer une opération entre deux tableaux, ils doivent nécessairement être de même dimensions.
Exemple : Testez les instructions suivantes :

```
u = [1 2 3 4 5 6] , v = [1 1 1 1]
u + v
u .* v
```

3.6 - Transposition

La transposition d'un tableau consiste en inversant le rôle des lignes et colonnes. Elle s'effectue avec le symbole ' (`apostrophe` ou `quote`)
Exemple : Tapez les instructions suivantes :

```
v = 1:8 , w = [4;5;6;7] , M = [1 2 3;4 5 6]
```

puis l'instruction `v'` puis l'instruction `w'` puis l'instruction `M'`.

Exercice 2 :

en utilisant les instructions vues précédemment, écrire (si possible de la manière la plus simple) les instructions pour créer les tableaux suivants :

$$M1 = \begin{pmatrix} 1 & 3 & 5 & 7 \\ 2 & 4 & 6 & 8 \\ 3 & 5 & 7 & 9 \\ 4 & 6 & 8 & 10 \\ 5 & 7 & 9 & 11 \\ 6 & 8 & 10 & 12 \\ 7 & 9 & 11 & 13 \end{pmatrix} \quad M2 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 5 & 5 & -2 & -2 & -2 \\ 5 & 5 & -2 & -2 & -2 \end{pmatrix} \quad M3 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 4 & 9 & 16 & 25 & 36 & 49 & 64 \end{pmatrix}$$

3.7 - Dimensions d'un tableau

A tout moment, il est possible de connaitre les dimensions d'un tableau avec les procédures `size` et `length`.
Exemple : Testez les instructions suivantes en observant le résultat après chacune d'elles :

```
v = 1:8 , w = [4;5;6;7] , M = [1 2 3;4 5 6] , t_vide = []
size(v) // donne les dimensions du tableau v : [nb_ligne nb_colonne]
size(w) // donne les dimensions du tableau w
size(M) // donne les dimensions du tableau M
size(M,1) // donne le nombre de lignes de M
size(M,2) // donne le nombre de colonnes de M
length(v) // donne le nombre de valeurs du tableau v
length(w) // donne le nombre de valeurs du tableau w
length(M) // donne le nombre de valeurs du tableau M
length(t_vide) // donne le nombre de valeurs du tableau t_vide
```

⇨ On peut définir un tableau vide `[]` avec zéro valeur.

4 - Fonctions

Scilab fournit un certain nombre de fonctions notamment les principales fonctions mathématiques (voir la partie *Elementary functions* de l'aide en ligne).
La plupart des fonctions peuvent s'appliquer aussi bien à une seule valeur qu'à un tableau de

valeurs.
Exemple : Testez les instructions suivantes :

```
sqrt(4) // racine carrée de 4
k = 0:12
sqrt(k) // les valeurs [sqrt(0),sqrt(1),sqrt(2),...,sqrt(11),sqrt(12)]
t = k * %pi / 12
cos(t) // les valeurs [cos(0),cos(pi/12),cos(2*pi/12),...,cos(pi)]
```

On peut à partir des opérateurs et fonctions de Scilab, créer ses propres fonctions.
Exemple : Pour calculer les valeurs de la fonction $f(x) = \dfrac{1}{1+x^2}$ pour les valeurs $x = 0$, $x = 0,5$, $x = 1$, $\ldots x = 9,5$ et $x = 10$, tapez les instructions suivantes :

```
x = 0:0.5:10
y = (1) ./ (1 + x .^ 2)
```

On aimerait pouvoir définir la fonction $f(x)$ puis l'utiliser à l'aide de l'instruction `y=f(x)`.

Scilab permet à l'utilisateur de créer ses propres fonctions à l'aide de l'instruction `deff`, et éventuellement de les sauvegarder sous forme de fichier pour réutiliser ultérieurement.
Exemple : Avec l'éditeur de texte, créez le fichier suivant en le nommant `ex_fct1.sce`

```
// definition de la fonction f
deff("y = f(x)" , "y = (1) ./ (1 + x .^ 2)");

//----- utilisation de la fonction f -----

// calcul de f(x) pour x=0  x=0,5  x=1  ... x=9,5  et  x=10
x = 0:0.5:10
y = f(x)
disp(y)

// calcul de f(t) pour 1000 valeurs de t équiréparties entre -1 et 2
t = linspace(-1,2,1000)
z = f(t)
disp(z)
```

puis exécutez le script `ex_fct1.sce`.
⇨ il est important d'utiliser les opérations terme à terme (`.*`, `./`, `.^`) pour pouvoir utiliser la fonction avec un tableau de valeurs.
⇨ une fois la fonction définie, on peut l'utiliser avec n'importe quelle variable (pas nécessairement avec des variables ayant les mêmes noms que dans la définition de la fonction).
⇨ dans le cas d'un tableau avec un grand nombre de valeurs, Scilab demande à l'utilisateur de continuer ou non l'affichage du tableau dans la console.

Une ou plusieurs fonctions peuvent être définies dans un fichier-script puis être utilisées dans un autre fichier-script Scilab.
Exemple : Avec l'éditeur de texte, créez le fichier suivant en le nommant `mes_fonctions.sci`

```
// definition de la fonction f1
deff("y = f1(x)" , "y = (1) ./ (1 + x .^ 2)");

// definition de la fonction f2
deff("y = f2(t)" , "y = (t+1) .^ 2");
```

puis créez le fichier suivant en le nommant `ex_fct2.sce`

```
// chargement du contenu du fichier nommé mes_fonctions.sci
exec("mes_fonctions.sci", -1);

// calcul de f1(x) pour x=0  x=0,5  x=1  ... x=9,5  et  x=10
x = 0:0.5:10
y = f1(x)
disp(y)

// calcul de f2(t) pour 100 valeurs de t entre -10 et 10
t = linspace(-10,10,100)
z = f2(t)
disp(z)
```

puis exécutez le script `ex_fct2.sce`.

⇨ par convention, les fichiers contenant uniquement des définitions de fonctions ont un nom avec le suffixe `.sci` alors que les scripts Scilab ont un nom avec le suffixe `.sce`

Exercice 3 :

ajouter au fichier nommé `mes_fonctions.sci` la définition de la fonction $y = f_3(x) = \exp(\sqrt{x/10})$ puis écrire un script Scilab nommé `ex_fct3.sce` qui permet de calculer et afficher les deux vecteurs $u = f_3(t)\cos(t)$ et $v = f_3(t)\sin(t)$ pour 1000 valeurs de t entre 0 et 10.

5 - Programmation

Scilab dispose d'un langage avec instructions structurées afin d'écrire des programmes complexes. Le langage est de type *impératif* (comme par exemple les langages C, Java, Ada).

5.1 - Variables et affectation

On peut stocker différentes valeurs (ou tableaux de valeurs) dans des variables, une variable étant identifiée par un nom, nom formé d'une lettre suivie éventuellement de lettre(s) et/ou de chiffre(s).

En Scilab, une variable se définit *à la volée* en lui affectant le résultat d'une expression.

Affectation

Notation algorithmique	Notation Scilab
variable ← *expression*	*variable* = *expression*

5.2 - Entrée-sortie

L'instruction `disp` permet l'affichage d'une variable ou d'une expression.
L'instruction `input` permet à l'utilisateur d'entrer une valeur, un tableau ou une chaîne de caractères.
Exemple : créez le fichier suivant en le nommant `ex_entree_sortie.sce`

```
n = input("Entrer un entier : ")
mprintf("n   = %d\n",n)
mprintf("n*n = %d\n",n*n)

t = input("Entrer un tableau de valeurs (entre crochets) : ")
disp(t)
```

puis exécutez le script `ex_entree_sortie.sce`.
⇨ dans certains cas, à la place de l'instruction `disp`, on utilisera l'instruction `mprintf` (correspondant à l'instruction `printf` du langage C) pour écrire des nombres (entiers ou réels).

Entrée-sortie

Notation algorithmique	Notation Scilab
lire (*variable*)	`input(`*variable*`)` `input(`*texte*`,`*variable*`)`
ecrire (*variable*)	`disp(`*variable*`)`
ecrire (*valeur_entiere*)	`mprintf("%d\n",`*valeur_entiere*`)`
ecrire (*valeur_reelle*)	`mprintf("%f\n",`*valeur_reelle*`)`

5.3 - Test

L'instruction `if` permet d'exécuter une suite d'instructions si et seulement si une expression est vraie.

Test simple

Notation algorithmique	Notation Scilab

si *expression_booleenne* **alors**
instructions
fin_si
	`if` *expression_booleenne* `then`
	instructions
	`end`

si *expression_booleenne* **alors**
instructions
sinon
instructions
fin_si
	`if` *expression_booleenne* `then`
	instructions
	`else`
	instructions
	`end`

On peut aussi enchâiner plusieurs tests à la suite :

Test multiple

Notation algorithmique	Notation Scilab

si *expression_booleenne* **alors**
instructions
sinon_si *expression_booleenne* **alors**
instructions
⋮
sinon
instructions
fin_si
	`if` *expression_booleenne* `then`
	instructions
	`elseif` *expression_booleenne* `then`
	instructions
	⋮
	`else`
	instructions
	`end`

Exemple : créez le fichier suivant en le nommant `ex_test.sce`

```
n = input("Entrer un entier n : ")
disp(n)


if n>2 then
    disp("n est supérieur à 2")
end

if n==0 then
```

```
    disp("n est nul")
elseif n>0 then
    disp("n est strictement positif")
else
    disp("n est strictement négatif")
end
```

puis exécutez le script `ex_test.sce`.

➪ les opérateurs de comparaison sont :

< inférieur strictement à	> supérieur strictement à	== égal à
<= inférieur ou égal à	>= supérieur ou égal à	<> différent de

➪ les opérateurs booléens sont :

& ET	| OU	~ NON

5.4 - Boucle conditionnelle

L'instruction `while` permet de répéter une suite d'instructions tant qu'une expression booléenne est vraie.

Boucle conditionnelle

Notation algorithmique	Notation Scilab
tant_que *expression_booleenne* **faire** *instructions* **fin_tant_que**	`while` *expression_booleenne* *instructions* `end`

Exemple : créez le fichier suivant en le nommant `ex_boucle1.sce`

```
// définir deux entiers a et b tels que 0 < a < b
a = 48; b = 66;
mprintf("a = %d , b = %d \n", a, b)
while a>0
    r = a;
    a = pmodulo(b,a);
    b = r;
end
mprintf("le pcgd de a et b est %d\n", b)
```

puis exécutez le script `ex_boucle1.sce`.

5.5 - Boucle inconditionnelle

L'instruction `for` permet de répéter une suite d'instructions pour un ensemble de valeurs.

Boucle inconditionnelle

Notation algorithmique	Notation Scilab
pour *variable* **appartenant à** *tableau_valeurs* **faire** *instructions* **fin_pour**	`for` *variable* = *tableau_valeurs* *instructions* `end`

Souvent, on l'utilise avec la syntaxe suivante :

Notation algorithmique	Notation Scilab
pour *variable* **de** *v_min* **à** *v_max* **faire** *instructions* **fin_pour**	`for` *variable* = *v_min*:*v_max* *instructions* `end`

et les instructions sont exécutées pour *variable* = *valeur_min*, *variable* = *valeur_min*+1, ..., jusqu'à *variable* = *valeur_max*

Exemple : créez le fichier suivant en le nommant `ex_boucle2.sce`

```
disp("Exemple 1")
for i = 1:10
    disp(i)
end

disp("Exemple 2")
tab_v = [2.4 7.4 8 3.1 9.5 0.1]
somme = 0;
for i = tab_v
    disp(i)
    somme = somme+i;
end
mprintf("La somme des elements de v est %f\n", somme);
```

puis exécutez le script `ex_boucle2.sce`.

5.6 - Commentaire

En Scilab, un commentaire correspond à la partie d'une ligne se trouvant après deux caractères `//` .

Exercice 4 :

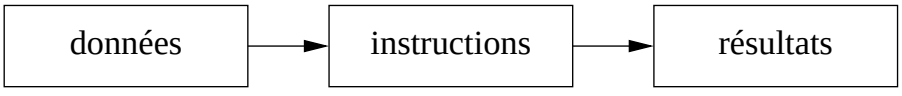
écrire un script Scilab nommé `ex_suite.sce` et qui effectue les opérations suivantes :
- demande à l'utilisateur d'entrer une valeur positive a
- calcule le vecteur u formé de 10 valeurs de la manière suivante :

$$u(1) = \frac{a+1}{2} \quad \text{et} \quad u(n) = \frac{a}{2\,u(n-1)} + \frac{u(n-1)}{2} \text{ pour } n \text{ variant de 2 à 10}$$

- affiche les valeurs du vecteur u ainsi que les valeurs au carré du vecteur u.

5.7 - Fonction générale Scilab

Une *fonction générale* en Scilab généralise le concept de fonction mathématique, c'est à dire à partir d'un ensemble (éventuellement vide) de données, on exécute une suite d'instructions qui fournit un ensemble (éventuellement vide) de résultats.
On peut le schématiser ainsi :



En Scilab les ensembles de données / de résultats seront décrits par des listes de variables.

Sous-programme SCILAB

Notation Scilab

`function` [*liste_variables_resultats*] `=` *nom_sous_programme* `(`*liste_variables_donnees*`)`
instructions
`endfunction`

Exemple : avec l'éditeur de texte, créez le fichier suivant nommé `ex_statistiques.sce`, où une fonction calculant la moyenne et l'écart-type d'un tableau de valeurs est définie puis utilisée.

```
// fonction calculant la moyenne m et l'écart-type s
// d'un tableau de valeurs tv
function [m,s] = statistiques(tv)

  n = length(tv); // nombre d'éléments dans le tableau tv
  s1 = sum(tv);   // somme des éléments de tv
  s2 = sum(tv .^ 2);   // somme des éléments au carré de tv

  m = s1/n; // la moyenne
  s = sqrt(s2/n-m*m); // l'écart-type

endfunction

// calcul des statistiques pour le tableau t1
t1 = [1 5 7 2 3 1 3 2 7];
disp(t1);
[m1,s1] = statistiques(t1);
mprintf("moyenne = %f , ecart-type = %f\n", m1, s1);

// calcul des statistiques pour le tableau t2
t2 = input("Entrer un tableau de valeurs entre crochets : ");
[m2,s2] = statistiques(t2);
mprintf("moyenne = %f , ecart-type = %f\n", m2, s2);
```

6 - Graphique

Scilab dispose de fonctionnalités afin de tracer des graphiques à partir de données sous forme de tableaux de valeurs. En MAP101, on utilisera essentiellement l'instruction `plot` sous l'une des formes suivantes

`plot`(x,y)

`plot`(x,y,*options_graphiques*)

6.1 - Représentation de points du plan

Les deux premiers arguments x et y de l'instruction `plot` sont deux vecteurs définissant les coordonnées de points dans le plan, le vecteur x correspond à des abscisses et le vecteur y correspond à des ordonnées.

On peut alors représenter un ensemble de points (x_i, y_i), $1 \leq i \leq n$ du plan, en définissant un vecteur `x` $= [x_1\ x_2\ \ldots\ x_n]$ et un vecteur `y` $= [y_1\ y_2\ \ldots\ y_n]$, chaque vecteur contenant n valeurs.

Exemple : Pour représenter les 4 points $(2,1)$, $(0,0)$, $(3,-1)$ et $(4,2)$, d'abord définir les deux vecteurs pour les abscisses et les ordonnées :

```
x1 = [ 2  0  3  4]
y1 = [ 1  0 -1  2]
```

puis effectuer le tracé :

```
scf(); // creer une nouvelle fenetre graphique
plot(x1,y1,".") // option "." : tracé de points
```

⇨ on remarque que les limites du repère correspondent aux limites des données soit l'intervalle $[0,4]$ en abscisse et l'intervalle $[-1,2]$ en ordonnée, et certains points sont peu visibles.

Pour modifier les limites du repère, il suffit d'utiliser l'instruction `replot([xmin,ymin,xmax,ymax])`

Exemple : Testez les instructions suivantes en observant le résultat après chacune d'elles : `replot([-1 -2 5 3])` , `replot([-10 -10 10 10])` , `replot([0 0 4 4])`

On peut modifier le mode de tracé en modifiant le troisième paramètre de la procédure `plot` qui est une option de tracé sous forme d'une chaîne de caractères.
Exemple : Testez les instructions suivantes en observant le résultat après chacune d'elles et observez notamment le résultat visuel des différentes options :

```
scf(), plot(x1,y1,"g-"), replot([-3 -2 5 3])
scf(), plot(x1,y1,"k--"), plot(x1,y1,'ro'), replot([-3 -2 5 3])
scf(), plot(x1,y1), plot(x1,y1,'c.'), replot([-3 -2 5 3])
```

⇨ on peut effectuer différents tracés dans une même fenêtre graphique en effectuant plusieurs instructions `plot`.
Exemple : Rajoutez les instructions suivantes :

```
x2 = [-1  0  1  0 -1]
y2 = [ 0  1  0 -1  0]
plot(x2,y2,'c:'), plot(x2,y2,'m*')
```

⇨ les points des vecteurs `x2` et `y2` correspondent aux sommets d'un carré, alors que la représentation graphique ne donne pas nécessairement un carré (on voit plutôt un losange). Pour avoir une représentation graphique plus juste, il faut faire en sorte que le repère soit normalisé (même échelle en abscisse et en ordonnée).
Exécutez l'instruction suivante :

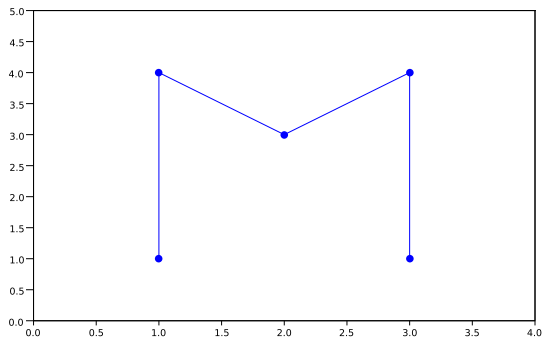
```
set(gca(),"isoview","on")
```

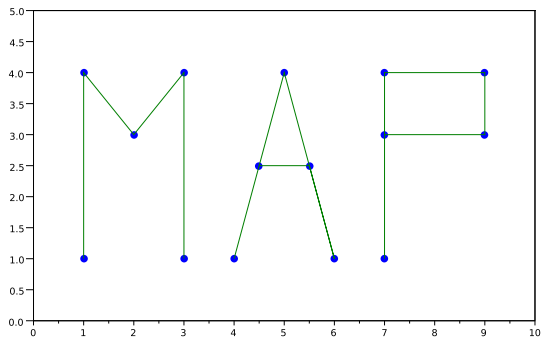
⇨ pour plus d'info sur l'instruction `plot`, tapez l'instruction `help plot` .

⇨ pour supprimer toutes les fenêtres graphiques, utiliser l'instruction `xdel(winsid())` .

Exercice 5 :

écrire les instructions Scilab afin d'obtenir les deux figures suivantes :





6.2 - Tracé de courbes représentatives de fonctions

Dans ce paragraphe, nous allons voir comment tracer le graphe $y = f(x)$ d'une fonction f dont on connait l'expression en fonction de x.

Le but de cette séance est que vous fassiez votre propre formulaire avec les représentations graphiques des principales fonctions usuelles.

Exemple : : tracer le graphe de $f(x) = x^2$. Le domaine de définition de f étant $\mathbb{R}$, on ne peut pas représenter le graphe pour toutes les valeurs x de $\mathbb{R}$ mais seulement pour un intervalle $I = [a, b]$. De plus, on ne peut pas calculer la fonction f sur toutes les valeurs de l'intervalle I (car il y en a une infinité), mais seulement avec un nombre fini de valeurs entre a et b

Créer un fichier script Scilab nommé `trace_x_puissance_2.sce` avec les instructions suivantes :

```
a = -2; b = 2; // les bornes de l'intervalle I
x = linspace(a,b,10); // choisir 10 valeurs équiréparties entre a et b
deff("y = f(x)","y = x .^ 2"); // définir la fonction
y = f(x); // calculer les valeurs y correspondant aux valeurs de x
scf(); // ouvrir une fenetre graphique
plot(x,y,'.');
```

et exécutez ce script. Seuls les 10 points du graphe correspond aux 10 valeurs du tableau `x` sons affichés.

Pour avoir un tracé continu, il faut relier les plans entre eux : modifiez le script précédent en remplaçant l'instruction `plot(x,y,'.')` par `plot(x,y,'-')` et ré-exécutez-le. Le graphique présente une courbe continue mais avec des points anguleux.

Pour avoir une courbe plus lisse, il faut augmenter le nombre de valeurs dans le tableau `x` : modifiez le script précédent en remplaçant l'instruction `x = linspace(a,b,10)` par `x = linspace(a,b,10` et ré-exécutez-le.

Ensuite, on peut éventuellement améliorer la représentation : rajouter à la fin du script les instructions suivantes :

```
replot([-2,-1,2,5]); // modifier les bornes du repère
xtitle("f(x) = x^2") // titre du graphique
axes = gca(); // le repère-axes graphique
axes.x_location = "origin"; // repère-axes passant par l'origine
axes.y_location = "origin";
axes.box = "off"; // supprimer la boite englobant le repère-axes
axe.isoview = "on"; // normaliser le repère
```

Exercice 6 :

Cet exercice fera l'objet d'un compte-rendu noté.

Le but de cet exercice consiste en la création de votre propre formulaire avec les graphiques des fonctions usuelles.

Pour cela, il faut compléter le script précédent afin de tracer dans des fenêtres séparées les courbes représentatives des différentes fonctions.

Ensuite il faut rassembler les figures des différentes fenêtres graphiques dans un document de type *triatement de texte* : démarrez sous Windows l'application *LibreOffice*, par le menu *Fichier*, créez un nouveau document de type texte. Indiquez au début du document vos noms et prénoms.

Ensuite, faites un copier-coller des différentes figures dans ce document : pour chaque fenêtre graphique faite avec Scilab, copiez le contenu dans le presse-papier (menu *Fichier*, item *Copier dans le presse-papier*) puis allez dans votre document texte *LibreOffice* et collez la figure à l'endroit voulu. Vous pouvez ensuite réduire la taille de la figure si vous le

souhaitez.

Une fois votre formulaire terminé, sauvegardez-le puis exportez-le au format PDF (menu *Fichier*, item *Exporter au format PDF*) puis envoyez ce fichier PDF par e-mail à votre enseignant de TP en indiquant comme sujet du message : `[MAP101] - Compte-rendu 1 -` *noms du binôme*