Chapitre 1

Première année, Partie 1

1.1 La fenêtre `Scilab`, les objets `Scilab`

Lorsqu'on tape une instruction dans la ligne de commande (ligne indiquée par $--\ >$ dans la fenêtre **`Console Scilab`**), celle ci est exécutée dès que la ligne est validée par la touche entrée $\boxed{\hookleftarrow}$.

La réponse apparaît à la ligne suivante sous le nom ***ans*** (pour answer).

La variable **`ans`** peut être réutilisée dans un calcul comme dans l'exemple suivant :

```
-->1+2
 ans  =
    3.
-->ans+5
 ans  =
    8.
```

Une instruction validée ne peut plus être modifiée.
Cependant on peut rappeler une instruction dans la ligne de commande en utilisant les touches $\boxed{\uparrow}$ ou $\boxed{\downarrow}$. **Essayez ! ! !**

1.1.1 Les nombres

Les nombres réels

Les nombres réels reconnus par **`Scilab`**, sont des nombres décimaux composés de 16 *chiffres significatifs* qui ne sont pas tous affichés à l'écran.
Pour modifier l'affichage, ce qui n'est pas très utile, on peut régler le menu ***préférences*** ou chercher dans l'aide `format` pour voir les divers choix.

Dans la suite, on écrira en général *valeur de* x au lieu de *valeur approchée de* x *avec une précision relative de l'ordre de* 10^{-16}.

Cette locution est en théorie incorrecte, mais bien pratique [1].

Un nombre réel se tape comme sur toute calculatrice en mode numérique (on utilise le point-décimal anglo-saxon et pas la virgule) :
`9.44` représente le nombre $9,44$

Dans la suite, comme la virgule sert de séparateur dans les suites de nombres, pour éviter les ambiguités de lecture, nous écrirons partout les nombres "à virgule" avec un point décimal.

Les nombres peuvent être écrits avec des puissances de 10 :

`96.57567e+3` et `96.57567D+3` représentent 96.57567×10^3 ou encore 96575.67 ;
notez qu'il n'y a pas d'espace entre `7` et `e` ni entre `7` et `D` .

`3.14e-3` et `3.14D-3`représentent 3.14×10^{-3} ou encore 0.00314 ;
notez qu'il n'y a pas d'espace entre `4` et `e` ni entre `4` et `D`.
En pratique, `d, D, e, E` sont acceptés.

L'expression `%e` est [une valeur approchée à 10^{-16} près de] l'unique nombre réel e tel que $\ln(e) = 1$.
Le nombre $3e$ se tape `3*%e` : ne pas oublier $\boxed{*}$ entre 3 et `%e`.

L'expression `%pi` est [une valeur approchée à 10^{-16} près de] du nombre réel π.
Le nombre 2π se tape `2*%pi` : ne pas oublier $\boxed{*}$ entre 2 et `%pi`.

Remarque
Le codage, interne à la machine, des nombres réels se fait sur un nombre fixe d'octets [2].
Cette limitation a plusieurs conséquences.

• Il existe un plus grand nombre reconnu par le logiciel.
Pour la version `5.4`, ce nombre est de l'ordre de 10^{308}, (à comparer par exemple avec la dette privée US qui n'est (été 2013) que de l'ordre de 10^{14}\$).
Lorsque, au cours d'un calcul, ce nombre est dépassé, il y a message d'erreur.

•• Il existe un plus petit nombre strictement positif reconnu par le logiciel.
Ce nombre ε est de l'ordre de 10^{-308}.
Si au cours d'un calcul, **Scilab** rencontre un nombre $x \in]-\varepsilon, \varepsilon[$, ce nombre x est remplacé par 0.

••• Si a est un nombre réel non nul et que x est un réel tel que $-2.10^{-16} < \dfrac{x}{a} < 2.10^{-16}$ alors, pour **Scilab**, le résultat de l'addition $a + x$ est égal à a, ce qui revient à dire que x est négligé devant a.

Travaux pratiques 1
Tapez dans la ligne de commande :
```
1+1.e-16
ans-1
```

1. Pour avoir une idée intuitive de ce que représente une précision relative de l'ordre de 10^{-16}, on peut remarquer que cela revient à mesurer la production annuelle de combustibles fossiles au gramme près.

2. Un octet est une suite de 8 bits (***bi**nary digi**t***).
Un bit est un des deux symboles 0 ou 1. Il y a donc $2^8 = 256$ octets différents.

On constate que 10^{-16} est négligé devant 1.

Tapez dans la ligne de commande :
`1+2.e-16`
`ans-1`
On constate que le résultat du calcul $(1+2.10^{-16})-1$ n'est pas exactement égal à 2.10^{-16}.
Ceci est dû aux erreurs d'arrondi dans le calcul.

Les nombres complexes : (ECS)

Le symbole `%i` représente le nombre complexe i.

On tape un nombre complexe quelconque sous la forme `x+y*%i` où `x` et `y` sont des réels.
Attention : ne pas oublier le symbole [*].
La réponse de **Scilab** est écrite sans `%` ni $*$.

Affichage à l'écran

Si on ne veut pas que **Scilab** écrive le résultat d'une instruction à l'écran, on termine la ligne par [;]. On dit que [;] est un *inhibiteur d'affichage* [3].

Les opérations sur les nombres

Les quatre opérations sont comme d'habitude : [+] [−] [*] [/].

Elles fonctionnent avec l'ordre de priorité habituel, ce qui nécessite l'usage convenable des **parenthèses**.

Attention :

$\frac{1}{2\times 3}$ se tape `1/(2*3)` ou `1/2/3`

$\frac{1}{2}\times 3$ se tape `1/2*3`.

1.1.2 Les fonctions de la variable réelle

Les fonctions d'une variable réelle au programme sont :

la fonction puissance : `2 ^ 3` calcule 2^3 ; (touche accent circonflexe)

le logarithme népérien : `log(10)` calcule $\ln(10)$;

l'exponentielle : `exp(-1)` calcule e^{-1} ;

le sinus : `sin(%pi/4)` calcule $\sin(\frac{\pi}{4})$, c'est-à-dire $\frac{\sqrt{2}}{2}$;

le cosinus : `cos(0.01)` calcule $\cos(0.01)$;

la valeur absolue :
si x est un nombre réel, `abs(x)` donne la valeur absolue de x,
(ECS) si x est un nombre complexe, `abs(x)` donne le module de x ;

la partie entière : si x est un nombre réel, `floor(x)` donne la partie entière de x, c'est-à-dire le plus grand entier inférieur ou égal à x.

3. Supprimer l'affichage libère le processeur pour les calculs et accélère l'exécution des programmes.

la racine carrée : `sqrt(421)` calcule $\sqrt{421}$;

Remarque
Sauf pour la fonction puissance, il est obligatoire de mettre des parenthèses.

> Aucune autre fonction de variable réelle n'est explicitement au programme de première année.

Remarque
En cas de division par zéro, ou si le résultat dépasse le plus grand nombre reconnu par **Scilab**, la réponse est un message d'erreur.
Selon le cas et le réglage des préférences, le message d'erreur prend une des formes suivantes :
`!--error 27 Division par zéro` , `Inf` ($+\infty$), `-Inf` ($-\infty$), `NaN` (Not a Number), etc.

Attention (ECS)

Si z est un nombre complexe dont la forme exponentielle est $\begin{cases} z = Re^{i\alpha} \\ -\pi < \alpha \leq \pi \end{cases}$
la valeur renvoyée par l'instruction `sqrt(z)` est $\sqrt{R}.e^{i\ \alpha/2}$, unique nombre complexe d'argument dans $]-\frac{\pi}{2}, \frac{\pi}{2}]$ dont le carré est égal à z.

Vérifiez le en tapant par exemple `sqrt(-1)`.

Remarque
Pour Scilab, les fonctions ci-dessus sont définies sur $\mathbb{C}$ (ou $\mathbb{C}^*$ pour `log`).
La cohérence avec la tradition mathématique n'est que partielle.

Plus généralement, **nous invitons les étudiants à ne jamais utiliser, sans invitation exprès de l'énoncé, une expression qui n'est pas définie dans leur cours de mathématique :**
introduire une expression non définie dans le cours de mathématiques comme `(-1)^(1/2)` ou `log(%i)`, même si **Scilab** lui donne un sens, ne leur rapporterait que les pires ennuis aux concours.

1.1.3 Le générateur de nombres aléatoires `rand`

La fonction `rand` renvoie un nombre au hasard selon une loi de probabilité définie à l'avance.
Lorsqu'on tape `rand` dans une instruction, on dit qu'on *appelle* la fonction `rand`.

Loi uniforme

`rand("uniform")` (ou l'instruction équivalente `rand('uniform')`) indique que chacune des instructions `rand()` qui suit donnera un nombre au hasard dans $[0, 1]$ en suivant une loi uniforme sur $[0, 1]$, loi traditionnellement notée $\mathcal{U}([0, 1])$.
Cela signifie que pour tous réels $a, b \in [0, 1]$ avec $a < b$, la probabilité pour que le nombre renvoyé par chacune des instructions `rand()` soit entre a et b est égal à $b - a$.
On dit que chaque appel `rand()` *simule* une variable uniforme de loi $\mathcal{U}([0, 1])$.

Des appels successifs de la fonction `rand()` sont indépendants [4].

Loi normale

`rand("normal")` (ou l'instruction équivalente `rand('normal')`) indique que chacune des instructions `rand()` qui suit donnera un nombre au hasard dans $\mathbb{R}$ en suivant une loi normale, ou loi de Laplace-Gauss, de paramètres $(0, 1)$, loi traditionnellement notée $\mathcal{N}(0, 1)$.
Cela signifie que pour tous réels a, b avec $a < b$, la probabilité pour que le nombre renvoyé par chacune des instructions `rand()` soit entre a et b est $\frac{1}{\sqrt{2\pi}} \int_a^b \exp(\frac{-t^2}{2})\, dt$.

On dit que chaque appel `rand()` simule une variable normale de loi $\mathcal{N}(0, 1)$.

Des appels successifs de la fonction `rand()` sont indépendants.

Attention
Si, lors d'une session **Scilab**, on ne précise pas quelle loi de probabilité doit suivre `rand`, c'est par défaut la loi uniforme sur $[0, 1]$.
Avec plusieurs systèmes d'exploitation, à la date d'impression de ce manuel, la fonction `rand` de la version 5.4.1 a un comportement surprenant, illustrée par le TP suivant :

Travaux pratiques 2
Quittez (***Fichier Quitter***), puis relancez **Scilab**. Tapez les instructions suivantes dans la ligne de commande et consignez par écrit les réponses de **Scilab**.

```
rand('uniform')
 rand()
 rand()
 rand()
```

Quittez **Scilab**.
Relancez **Scilab** puis tapez de nouveau les 4 lignes de commande au dessus.

Vous constatez que :
ou bien `rand()` renvoie toujours la même suite de nombres aléatoires,
ou bien `rand()` renvoie, à un décalage d'une unité près, toujours la même suite de nombres aléatoires.

On évite ce comportement, inopportun pour la plupart des applications, en commençant toute session faisant appel à `rand` par [5] : `rand("seed", getdate('s'))` ou toute instruction équivalente.

Remarque
Avec les dernières versions de **Scilab**, les parenthèses vides ne sont pas toujours obligatoires. On peut écrire `rand` au lieu de `rand()`.

4. En fait, on ne peut pas **démontrer** qu'un algorithme réalise des tirages indépendants de nombres au hasard. On peut au mieux vérifier qu'il satisfait les critères nécessaires d'indépendance connus à ce jour.

5. En tapant `help rand` dans la ligne de commande, vous ouvrez une rubrique d'aide sur le générateur de nombres aléatoires `rand`.

1.2 Objets et commandes élémentaires

1.2.1 Variables réelles

Qu'est qu'une variable ?

Intuitivement, une **variable numérique** peut être considérée comme une fiche composée de deux parties :
— son nom qui est une suite formée de lettres, de chiffres, et du caractère "souligné", commençant par une lettre.
— sa valeur (ou affectation) qui est un nombre réel ou complexe.

Remarque
Dans le réglage des préférences par défaut, **Scilab** est sensible à la casse *(case dependant)*, c'est-à-dire qu'il distingue les majuscules des minuscules.

Comment créer une variable ?

Pour créer une nouvelle variable numérique, on écrit son nom à gauche de [=], puis on écrit sa valeur ou le procédé de calcul de sa valeur à droite.

Considérons les deux instructions :

```
a=20.6
X=2 ^(1/2)
```

La première crée la variable a, c'est-à-dire la variable dont le nom est a, et lui affecte le nombre 20.6 ; cette instruction s'appelle l'**affectation**.
La seconde crée la variable X à laquelle elle affecte $\sqrt{2}$.
La fenêtre *Navigateur de variables* contient maintenant les variables a et X.

Comment supprimer une ou plusieurs variables ?

Taper l'instruction `clear a` dans la ligne de commande supprime la variable a.
Taper l'instruction `clear` dans la ligne de commande supprime toutes les variables.
Quitter **Scilab** supprime toutes les variables.

Comment modifier la valeur d'une variable ?

Pour modifier la valeur d'une variable numérique qui existe déjà, on écrit son nom à gauche de [=], puis on écrit sa nouvelle valeur à droite.
Dans l'exemple suivant, la seconde ligne efface l'ancienne affectation ($5e$) de a, *qui est définitivement perdue*, et la remplace par 2.718.

```
a=5*%e
a=2.718
```

Comment utiliser une variable dans un calcul ?

Pour utiliser (la valeur d') une variable dans un calcul, on fait comme en mathématiques, on écrit son nom.
Considérons l'exemple suivant :

```
a=2.18
b=a+2
b=log(b)
```

La seconde instruction crée la variable b, et lui affecte la valeur de a augmentée de 2, c'est-à-dire 4.18.
Le troisième instruction prend la valeur de la variable b (ici 4.18), calcule $\ln(4.18)$, puis affecte cette valeur à b.
L'ancienne valeur de b est perdue.

fonction `disp`

L'instruction `disp(objet)` écrit la valeur de `objet`.
L'instruction

`disp(` *objet 1*`,` *objet 2*`, ...,` *objet n* `)`

écrit l'un en dessous de l'autre les valeurs de *objet n*, . . ., *objet 2*, *objet 1* .

Testez les exemples suivants et éventuellement quelques autres

```
disp(3+4)
a=8; disp(a*a);
a=8; b=a*(a-1); disp(a,b);
```

On n'utilise pas la fonction `disp` dans la ligne de commande car il est plus économique d'écrire `a` (sans point virgule) que `disp(a, "a=")`.
Nous verrons plus loin qu'on l'utilise systématiquement dans les programmes.

Variables chaînes de caractères (exemples à tester)

Les instructions

```
a='bonjour'
b="Bonsoir"
C='Morgen"
```

définissent des chaînes de caractères, c'est-à-dire des suites ordonnées de lettres.

Pour faire écrire la valeur de `a`, on demande : `disp(a)`.

Pour obtenir une chaîne de caractères contenant le caractère apostrophe ou le guillemet anglo-saxon, on redouble l'apostrophe ou le guillemet anglo-saxon, comme dans l'exemple suivant.

```
c= 'aujourd''hui'
d="l"'isle au tresor"
d="l''isle au tresor"
```

Pour concaténer (juxtaposer) des chaînes de caractères, on utilise +.

```
a='Time';  b= ' is ';  c= 'gold';
a=a+b+c
```

Lorsqu'une suite de nombres ou une opération est entre apostrophes, elle est considérée comme une suite de lettres et n'est pas évaluée. Testez les instructions suivantes :

```
disp(3*4, " 3*4 = " )
disp(3*5, " 3*4 = " )
```

Le programme EC utilise peu les chaines de caractères.

1.2.2 Vecteurs-ligne, vecteurs-colonne

Un vecteur-ligne est une suite, éventuellement vide, écrite horizontalement, de nombres qu'on appelle ses coefficients ou ses composantes.
Un vecteur-colonne est une suite, écrite verticalement, éventuellement vide, de nombres qu'on appelle ses coefficients ou ses composantes.
Le nombre de ses coefficients est aussi appelé sa longueur ou sa taille.
Lorsque `A` est un vecteur (ligne ou colonne), l'instruction `length(A)` renvoie la longueur de `A`.

On définit un vecteur-ligne en indiquant la suite des nombres qui le composent, séparés par des virgules, le tout entre des crochets carrés.
On définit un vecteur-colonne en indiquant la suite des nombres qui le composent, séparés par des points-virgule, le tout entre des crochets carrés.

Exemples

L'instruction `a=[2,5,sqrt(3),-2 -1/2]` crée le vecteur-ligne `a`
de valeur : $[2 \quad 5 \quad \sqrt{3} \quad -2 \quad -0.5]$

L'instruction `B= [5;sqrt(3);2]` crée le vecteur-colonne `B` de valeur : $\begin{bmatrix} 5 \\ \sqrt{3} \\ 2 \end{bmatrix}$

L'instruction `v=[ ]` crée un vecteur vide.
Trouvez la longueur de `v` en tapant `length(v)`.
Trouvez de même la longueur des vecteurs `a` et `B`.

Un coup d'oeil au navigateur de variables permet de voir que :
la variable `a` est de *taille* 1×5, c'est-à- dire 1 ligne et 5 colonnes ,
la variable `B` est de *taille* 3×1, c'est-à- dire 3 lignes et 1 colonne.
Quelle est la taille de `v` ?

Remarque :
Le terme *dimension* utilisé par le navigateur de variables de **Scilab** a une autre signification dans le cours de mathématiques.
Pour limiter les risques de confusion, nous utiliserons les termes *longueur* ou *taille* aussi utilisés par **Scilab**.

Accéder aux coefficients

Le premier coefficient d'un vecteur (ligne ou colonne) `a`, de longueur $n \geq 1$, est `a(1)`, le second `a(2)` et ainsi de suite jusqu'à `a(`n`)`.
Chacun de ses coefficients est une variable numérique.

Si k est un entier plus grand que la longueur n du vecteur `a`, écrire `a(`k`)`, à droite du symbole d'affectation $\boxed{=}$ ou dans une fonction, amène une erreur.