Informatique et Algorithmique – Chapitre 1

Découverte de Scilab

I Prise en main de Scilab

1) Un logiciel de calcul numérique

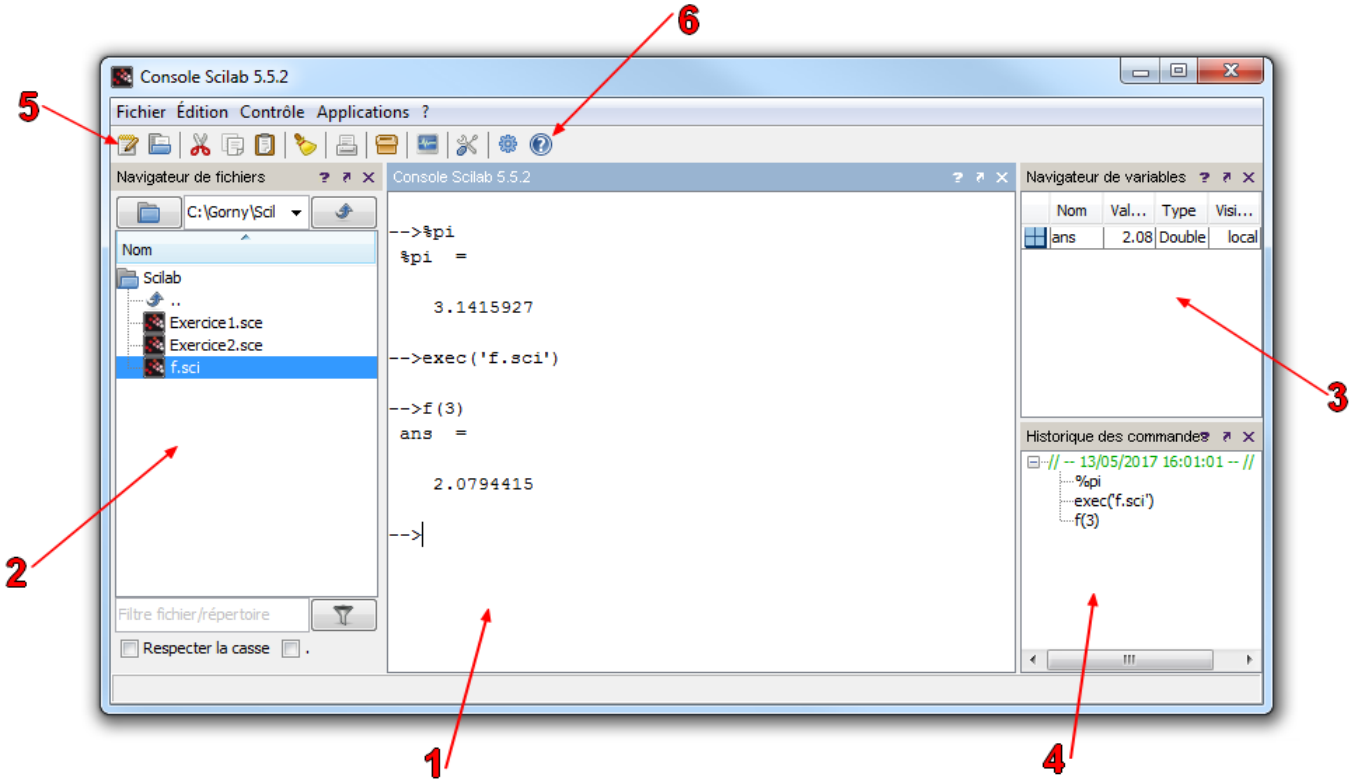
Le logiciel retenu pour la programmation dans le programme d'ECS est **Scilab**, abréviation de *Scientific Laboratory*. Il s'agit d'un logiciel open source gratuit de calcul numérique qui fournit un puissant environnement de développement pour les applications scientifiques et l'ingénierie.

Un logiciel de calcul numérique manipule des expressions numériques. C'est un outil de calcul scientifique, qui permet d'obtenir un résultat approché avec une grande précision. Notons la différence avec les logiciels dits de calcul formel qui manipulent des expressions symboliques et utilisent des expression exactes [1].

Vous pouvez télécharger Scilab gratuitement sur le site `https://www.scilab.org/fr`. Vous trouverez notamment sur ce site des aides et des tutoriels qui peuvent être de bons compléments à ce cours.

2) L'environnement de travail Scilab

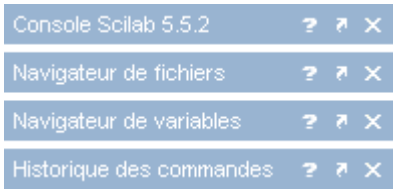
Lorsqu'on on lance Scilab, une fenêtre apparaît composée de plusieurs sous-fenêtres : la *console Scilab*, le *navigateur de fichier*, le *navigateur de variables* et l'*historique des commandes*.



1. Par exemple, un logiciel formel travaillera avec $\sqrt{2}$, alors qu'un logiciel de calcul numérique travaillera avec son approximation numérique $1,41421356237$

1. La **console Scilab** est la zone où l'on tape les commandes. Le symbole `-->` signifie que l'ordinateur attend une instruction. On peut alors saisir une commande et appuyer sur la touche [Entrée] pour que cette commande soit interprétée.
   - On peut écrire plusieurs instructions sur une même ligne en les séparant d'une virgule.
   - On peut faire suivre une commande par un point-virgule si l'on ne souhaite pas que le résultat de la commande s'affiche dans la console. On dit que l'exécution se fait *sans écho*.
   - Appuyer sur les touches [Ctrl] + [C] arrête l'exécution d'une commande.
   - L'usage des touches fléchées [↑] et [↓] fait apparaitre les commandes précédemment tapées.
2. Le **navigateur de fichier** est la zone où l'on peut choisir le répertoire courant de Scilab, c'est-à-dire le répertoire dans lequel se trouvent les fichiers avec lesquels on travaille [1]. On peut notamment les ouvrir en double-cliquant dessus.
3. Le **navigateur de variables** est la zone qui répertorie toutes les variables crées par l'utilisateur depuis l'ouverture de Scilab.
4. L'**historique des commandes** est la zone qui liste toutes les commandes que l'utilisateur a tapées dans la console Scilab.
5. L'icône permet d'ouvrir l'éditeur de texte **SciNotes** dans une nouvelle fenêtre. Cet outil permet de créer des scripts (des programmes, des fonctions, etc.) que l'on peut exécuter ensuite dans la console. Nous y reviendrons en détail dans le chapitre suivant.
6. L'icône permet d'ouvrir l'**Aide en ligne** dans une nouvelle fenêtre (on peut également taper la commande `help` dans la console). Cet outil permet de trouver une commande Scilab et de savoir comment l'utiliser. Pour obtenir des informations sur une commande en particulier, on peut taper dans la console `help` suivi du nom de la commande.

On peut personnaliser l'environnement de travail, notamment en modifiant la taille des fenêtres grâce aux doubles flèches qui apparaissent lorsque l'on place le curseur de la souris entre deux fenêtres. On peut également déplacer des fenêtres. Pour cela, on clique sur la fenêtre que l'on veut déplacer : une bande bleue [2] apparait. Il suffit alors de maintenir le chic gauche de la souris enfoncé pour déplacer la fenêtre à l'endroit voulu.



On peut également supprimer la fenêtre en cliquant sur la petite croix (attention la fermeture de la console provoque bien la fermeture de Scilab...). Enfin on peut insérer l'éditeur de texte SciNotes à l'espace de travail en employant la même méthode (c'est ce que l'on fera ultérieurement).

II Calculs sur les nombres réels

Scilab utilise la représentation des nombres réels en *virgule flottante* en suivant la norme IEEE-754. Hormis le nombre 0, Scilab travaille avec les réels compris entre environ $2,225\times10^{-308}$ et $1,798\times10^{308}$ et leurs opposés, avec une approximation décimale maximale de $\varepsilon \approx 2,22 \times 10^{-16}$.

Exemples :

```
-->10^308
 ans  =
    1.00D+308
```

```
-->10^309
 ans  =
    Inf
```

Les opérations arithmétiques élémentaires sur les réels se font à l'aide des symboles suivants :

+	addition	*	multiplication	∧	puissance
-	soustraction	/	division	**	puissance

1. Pour moi il s'agit de `C:\Gorny\Scilab\`
2. Elle est bleue sous Windows, mais noire sous Mac OS X ou Linux.

Exemples :

```
-->2+4,2-4
 ans  =
    6.
 ans  =
 - 2.
```

```
-->2*4,2/4
 ans  =
   8.
 ans  =
   0.5
```

```
-->2^4,2**4
 ans  =
   16.
 ans  =
   16.
```

 Les opérations en Scilab, comme en mathématiques, ont un ordre de priorité ($\wedge$ puis $*$ ou $/$ et enfin $+$ ou $-$) et on utilise donc également des parenthèses.

Exemples :

```
-->7*4-6, 7*(4-6)
 ans  =
   22.
 ans  =
 - 14.
```

```
-->5/7/2, (5/7)/2, 5/(7/2)
 ans  =
   0.3571429
 ans  =
   0.3571429
 ans  =
   1.4285714
```

```
-->3^4^2, 3^(4^2), (3^4)^2
 ans  =
   43046721.
 ans  =
   43046721.
 ans  =
   6561.
```

Certaines constantes sont prédéfinies dans Scilab (il s'agit bien sûr d'approximations) :

`%pi`	le nombre π
`%e`	la constante d'Euler $e = \exp(1)$

Scilab intègre également des fonctions prédéfinies :

`log`	la fonction logarithme népérien	`floor`	la fonction partie entière
`exp`	la fonction exponentielle	`cos`	la fonction cosinus
`sqrt`	la fonction racine carrée	`sin`	la fonction sinus
`abs`	la fonction valeur absolue	`tan`	la fonction tangente

Exemples :

```
-->%e**2, exp(2)
 ans  =
   7.3890561
 ans  =
   7.3890561

-->log(7.389)
 ans  =
   1.9999924
```

```
-->floor(-2.56)
 ans  =
 - 3.

-->sqrt(2), sqrt(-2)
 ans  =
   1.4142136
 ans  =
   1.4142136i
```

```
-->abs(log(0.5))
 ans  =
   0.6931472

-->%pi, cos(%pi), sin(%pi)
 %pi  =
   3.1415927
 ans  =
 - 1.
 ans  =
   1.225D-16
```

Comme on peut le remarquer sur ci-dessus, Scilab permet aussi de manipuler des (approximations des) nombres complexes grâce à la variable `%i` qui représente le complexe i.

Exemples :

```
-->%i, %i*%i, 1/(1+%i)
 %i  =
   i
 ans  =
 - 1.
 ans  =
   0.5 - 0.5i
```

```
-->exp(%i*%pi/4)
 ans  =
   0.7071068 + 0.7071068i

-->abs(%i)
 ans  =
   1.
```

La valeur d'un calcul affiché dans la console est un arrondi et non le contenu exact de la valeur conservée en mémoire. Pour afficher une valeur avec plus de précision, on peut changer le format d'affichage avec la fonction `format` (qui prend en argument un entier entre 2 et 25).

Exemples :

```
-->format()
 ans  =
    1.    10.

-->%pi, format(20); %pi
 %pi  =
    3.1415927
 %pi  =
    3.14159265358979312
```

```
-->sin(%pi),format(6),sin(%pi)
 ans  =
    0.00000000000000012
 ans  =
    0.000

-->format(26)
          !--error 999
format : Valeur erronée de l'argument n° 1 :
Doit être dans l'intervalle [2,25].
```

La commande `format()` affiche le format actuel. Dans l'exemple ci-dessus le `10.` signifie que l'affichage actuel utilise 10 caractères en comptant le signe. Mentionnons l'existence de la variable `%eps` qui renvoie la valeur de la précision machine ε.

III Création de variables par affectation

Nous avons déjà rencontré des variables dans les exemples précédents. Lorsque l'on effectue un calcul dans la console, le résultat de ce calcul se stocke dans une variable appelée `ans`.

Plus généralement, les variables permettent de stocker des données. Créer une variable par affectation consiste à associer une valeur à un nom. Si on dispose d'une valeur `expression` (qui est soit directement une valeur, soit une expression permettant le calcul d'une valeur), on utilise la syntaxe `nom=expression` pour affecter la valeur `expression` à la variable appelée `nom`.

S'il n'existe pas de variable appelée `nom`, alors cette instruction crée la variable `nom` et lui affecte `expression`. Si la variable `nom` existe déjà, alors cette instruction efface l'ancienne valeur de la variable et la remplace par la valeur `expression`. Pour connaître la valeur affectée à une variable existante, il suffit de taper son nom dans la console et de valider.

La commande `nom=expression;` permet d'affecter la valeur `expression` à la variable `nom` sans que le résultat ne s'affiche dans la console (pratique si l'on faut tourner un programme effectuant des milliers d'affectations). On dit que l'exécution de fait sans écho.

Exemples :

```
-->x
  !--error 4
Variable non définie : x

-->x=3; y=4; z=x*y+1
 z  =
    13.
```

```
-->4*x-5, x
 ans  =
    7.
 x  =
    3.
```

```
-->x=6*x-3; x
 x  =
    15.

-->t=ans/2+1
 t  =
    4.5
```

Remarques :

- ⚠ Le nom d'une variable ne doit pas commencer par un chiffre. Il peut être constitué d'au plus vingt-quatre caractères, chacun pouvant être une lettre sans accent, un chiffre ou un des symboles suivante : `_`, `#`, `!`, `?`, `$`. Scilab fait également la distinction entre minuscules et majuscules.
- Il est suggéré d'utiliser des noms courts et explicites pour les variables. Cela permettra de s'y retrouver plus facilement lorsqu'on reprendra un code ultérieurement. N'hésitez pas à ajouter des commentaires (avec le symbole //) pour expliquer le contenu d'une variable).
- ⚠ Le symbole = dans Scilab est appelé l'opérateur d'affectation. Il n'a pas le sens usuel du = mathématique. De plus il faut bien respecter la syntaxe : ce qui se trouve à gauche du symbole = est modifié ou créé par ce qui se trouve à droite du symbole =.

Comme on l'a vu plus haut, on peut affecter à une variable une expression contentant d'autres variables.... y compris elle même :

Exemple :

```
-->y=y**2+2*y+1//on remplace la valeur de a par la valeur de a^2+2a+1
 y  =
    25.
```

Quelques commandes utiles :

`who_user`	donne la liste des variables crées par l'utilisateur et les fonctions qu'il a appelées
`clear nom1,nom2`	efface les variables `nom1` et `nom2` de l'espace de travail
`clear`	efface toutes les variables crées dans l'espace de travail

On peut affecter différents types de données à des variables : des nombres bien sûr, mais aussi des vecteurs, matrices, chaines de caractères, booléens, etc. Nous y reviendrons en détail. Notons l'existence de la fonction `typeof` qui permet d'associer à chaque donnée son type.

Exemple :

```
-->x=2*%pi; typeof(x)
 ans  =
 constant
```

IV Variables booléennes et opérateurs de comparaison

Un booléen, en logique et en programmation informatique, est un type de variable à deux états : `T` (vrai) ou `F` (faux). En Scilab, les variables booléennes sont `%t` pour vrai et `%f` pour faux.

Scilab permet de faire des tests logiques avec les variables booléennes et les opérateurs logiques

`&`	opérateur et	`|`	opérateur ou	$\sim$	opérateur non

Exemples :

```
-->%f, typeof(%f)
 ans  =
  F
 ans  =
 boolean
```

```
-->~%t, ~%f
 ans  =
  F
 ans  =
  T
```

```
-->%t|%t, %t|%f
 ans  =
  T
 ans  =
  T
```

```
-->%t&%t, %t&%f
 ans  =
  T
 ans  =
  F
```

Scilab manipule des égalités et inégalités de réels permettant d'effectuer des tests logiques de comparaison. Les opérateurs de comparaison sont :

`==`	test d'égalité	`>`	supérieur strict	`<`	inférieur strict
`<>`	différent	`>=`	supérieur ou égal	`<=`	inférieur ou égal

Le résultat d'un test de comparaison est une variable booléenne.

Exemples :

```
-->5>=6
 ans  =
  F
```

```
-->log(%e)==1
 ans  =
  T
```

```
-->x=%pi/4; y=(x>0)&(x<1); y
 y  =
  T
```


Ne pas confondre l'opérateur d'égalité == et l'opérateur d'affectation =.

```
-->5=6
Attention : Utilisation obsolète de '=' à la place de '=='.
```

On peut également faire des opérations arithmétiques avec les variables booléennes. Dans ce cas elles sont automatiquement converties en réels : `%t` en 1 et `%f` en 0.

Exemple :

```
k=%f; 0.7^k*0.3^(1-k)
 ans  =
    0.3
```

Remarque : La précision de Scilab peut provoquer des erreurs dans les tests. Par exemple

```
sqrt(3)^2==3
 ans =
   F
```

Pour comprendre d'où provient l'erreur, étudions l'écart relatif entre `sqrt(3)`$\wedge$`2` et 3 et comparons la avec la précision ε de Scilab (stockée dans la variable `%eps`).

```
-->abs(sqrt(3)^2-3)<%eps
 ans  =
  F
```

```
-->abs(sqrt(3)^2-3)<3*%eps
 ans  =
  T
```

V Chaines de caractères

Une chaîne de caractère (*string* en anglais) est une suite de caractères alphanumériques. En Scilab, les chaînes de caractères sont délimitées par des apostrophes ' ou des guillemets ". On peut stocker une chaine de caractère dans une variable.

Exemple :

```
-->a="Hello"
 a  =
 Hello
```

Lorsqu'une chaîne de caractère contient le caractère ' ou le caractère ", il faut le doubler.

```
-->b='J'utilise Scilab'
       !--error 276
Opérateur, virgule ou point-virgule manquant.
```

```
-->b='J''utilise Scilab'
 b  =
 J'utilise Scilab
```

L'opérateur `+` permet de concaténer deux chaînes de caractères.

Exemples :

```
-->c='World'
 c  =
 World
```

```
-->a + c
 ans  =
 HelloWorld
```

```
-->a + ' ' + c + '!'
 ans  =
 Hello World!
```

 Lorsqu'une chaine de caractère est composée exclusivement de chiffres, il ne faut pas confondre l'opérateur + de la concaténation et celui de l'addition. De même on ne peut pas additionner un nombre et une chaine de caractère. Par contre on peut convertir une variable numérique en chaine de caractère avec la fonction `string`.

```
-->'20'+'17'
 ans  =
 2017

-->20+17
 ans  =
    37.

-->typeof('20')
 ans  =
 string
```

```
-->20+'17'
    !--error 144
Opération non définie pour les opérandes données.

-->x=%pi
 x  =
    3.1415927

-->"Une valeur approchée de pi est " + string(x) + '.'
 ans  =
 Une valeur approchée de pi est 3.1415927.
```

VI Vecteurs/listes

Scilab manipule les vecteurs (c'est-à-dire des listes ordonnées de réels, ou de booléens, ou de chaînes de caractère, etc.) que l'on peut également stocker dans une variable. Pour construire un vecteur par extension, on donne la liste des éléments du vecteur entre crochets et en les séparant de virgules. La commande `[]` construit une liste vide.

Exemples :

```
-->A=[1,2,3,4,5,6]
 A  =
    1.    2.    3.    4.    5.    6.

-->B=[%f,%t,%t,%f,%f]
 B  =
  F T T F F
```

```
-->C=['nord','sud','est','ouest']
 C  =
!nord  sud  est  ouest !

-->D=[]
 D  =
     []
```

Il existe d'autres manières de construire des vecteurs contenant des nombres : si `x` et `y` sont des réels, si `p` est un réel strictement positif et si `n` un entier strictement positif, alors

`x:y`	construit un vecteur constitué des nombres de `x` à `y` par pas de 1.
`x:p:y`	construit un vecteur constitué des nombres de `x` à `y` par pas de `p`.
`linspace(x,y,n)`	construit un vecteur constitué de `n` nombres entre `x` et `y`, régulièrement espacés.
`ones(1:n)`	construit un vecteur de taille `n` constitué uniquement de 1.
`zeros(1:n)`	construit un vecteur de taille `n` constitué uniquement de 0.
`rand(1:n)`	construit un vecteur de taille `n` constitué de nombres tirés *aléatoirement* dans $]0,1[$.

Exemples :

```
-->-2:5, 3:2
 ans  =
  - 2. - 1.    0.    1.    2.    3.
4.    5.
 ans  =
     []
-->2:0.1:3
 ans  =
    2.    2.1   2.2   2.3   2.4
2.5   2.6   2.7   2.8   2.9   3.
```

```
-->linspace(1,3,5), ones(1:4), zeros(1:5)
 ans  =
    1.    1.5   2.    2.5   3.
 ans  =
    1.    1.    1.    1.
 ans  =
    0.    0.    0.    0.    0.

 -->rand(1:6)
 ans  =
    0.4148104   0.2806498   0.1280058
0.7783129   0.2119030   0.1121355
```

Si `v` et `w` sont des vecteurs, si `x` est un réel et si `i` et `j` sont des entiers strictement positifs, alors

`length(v)`	renvoie la taille du vecteur `v`
`v(i)`	renvoie la $i^{\text{ième}}$ coordonnée du vecteur `v`
`v($)`	renvoie la dernière coordonnée du vecteur `v`
`v(i)=x`	remplace la $i^{\text{ième}}$ coordonnée du vecteur `v` par le réel `x`
`v(i)=[]`	supprime la $i^{\text{ième}}$ coordonnée du vecteur `v`
`v(i:j)`	renvoie un vecteur contenant les coordonnées de `i` à `j` de `v`
`v(i:j)=w`	remplace les coordonnées de `i` à `j` de `v` par les coordonnées de `w` (un vecteur de même taille que `v(i:j)`)
`v(i:j)=[]`	supprime les coordonnées de `i` à `j` de `v`
`[v,w]`	renvoie la concaténation des vecteurs `v` et `w`

Exemples :

```
-->v=[1,2,3,4,5,6]; v(2), v($)
 ans  =
    2.
 ans  =
    6.
```

```
-->v(7)
     !--error 21
Index invalide.

-->v(3)=7; v(4)=[]
    1.    2.    7.    5.    6.
```

```
-->v(4:5)=[-1,-2], v(2:3)=[]
 v  =
    1.    2.    7.  - 1.  - 2.
 v  =
    1.  - 1.  - 2.

-->v(1:2)
 ans  =
    1.    2.
```

```
-->v(3:2)
 ans  =
     []

-->v=[1,-1,-2]; w=[8,9]; [v,w]
 ans  =
    1.  - 1.  - 2.    8.    9.

-->v=[v,0]
 v  =
    1.  - 1.  - 2.    0.
```

Insistons sur la dernière commande de l'exemple : si on dispose d'un vecteur `v` et d'un nombre `x`, alors la commande `v=[v,x]` (resp. `v=[x,v]`) ajoute une nouvelle coordonnée au vecteur `v` contenant la valeur `x` en dernière (resp. première) position.

Comme pour les nombres, on peut faire des opérations (terme à terme) sur les vecteurs : si `v` et `w` sont des vecteurs de même taille, et si `x` est un réel non nul, alors

`v+w`	additionne terme à terme les vecteurs `v` et `w`
`v-w`	soustrait terme à terme le vecteur `w` au vecteur `v`
`x*v`	multiplie chaque coordonnée de `v` par `x`
`v/x`	divise chaque coordonnée de `v` par `x`
`v.*w`	multiplie terme à terme les vecteurs `v` et `w`
`v./w`	divise terme à terme le vecteur `v` par `w`
`v.^w`	effectue la puissance terme à terme du vecteur `v` par le vecteur `w`

Les fonctions usuelles (`log, exp, sqrt, abs, floor, cos, sin, tan`) s'appliquent également aux vecteurs coordonnée par coordonnée.

Exemples :

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

-->v+w, v-w, 2*v, w/2
 ans  =
    9.    9.    9.    9.
 ans  =
  - 7.  - 3.    1.    5.
 ans  =
    2.    6.    10.    14.
 ans  =
    4.    3.    2.    1.

-->v.*w,v./w, v.^w
 ans  =
    8.    18.    20.    14.
 ans  =
    0.125    0.5    1.25    3.5
 ans  =
    1.    729.    625.    49.
```

```
-->v+[1,2]
 !--error 8
Addition incohérente.

-->v*w
    !--error 10
Multiplication incohérente.

-->floor(log(w)), exp(v), sin(w)
 ans  =
    2.    1.    1.    0.
 ans  =
    2.7182818   20.085537   148.41316
1096.6332
 ans  =
    0.9893582 - 0.2794155 - 0.7568025
0.9092974
```

Enfin les opérateurs de comparaison et les opérateurs logiques s'appliquent aussi aux vecteurs de même taille. Les opérations se font terme à terme et le résultat est un vecteur de même taille contenant des variables booléennes.

Exemple :

```
-->u=[1,2,3,4]; v=[1,%pi,2,%e]; w=[%e,sqrt(2),0,5]; (v>=w)|(u>w)
 ans  =
  F T T F
```

Terminons par deux fonctions : si v est un vecteur constitué de booléens, alors

or(v)	retourne T si au moins un des éléments de v vaut T et retourne F sinon
and(v)	retourne T si tous les éléments de v ont la valeur T et retourne F sinon

Exemples :

```
-->v=[%t,%t,%f]; or(v), and(v)
 ans  =
  T
 ans  =
  F
```

Remarque : Nous avons uniquement vu les vecteurs lignes pour le moment. Nous verrons les vecteurs colonnes et plus généralement les matrices ultérieurement. Retenons pour l'instant la commande `u'` qui transforme le vecteur ligne u en un vecteur colonne (on dit que l'on transpose le vecteur u). Par exemple :

```
-->u=[1,3,5,7]; u'
 ans  =
    1.
    3.
    5.
    7.
```

```
-->u+u'
     !--error 8
Addition incohérente.

-->u*u'
 ans  =
    84.
```

 L'addition d'un vecteur ligne et d'un vecteur colonne n'a pas de sens et renvoie une erreur. La multiplication par contre fonctionne... nous verrons dans le chapitre sur les matrices ce que cela signifie.

VII Polynômes

Scilab permet également de définir et d'effectuer des opérations algébriques sur les polynômes. Soit v un vecteur Scilab non vide.

- La commande `poly(v,'X','c')` définit le polynôme dont le nom de l'indéterminée du polynôme est X et dont les coefficients sont les coordonnées du vecteur v (dans l'ordre croissant ; le coefficient constant étant la première coordonnée du vecteur).
- La commande `poly(v,'X')` définit le polynôme dont le nom de l'indéterminée du polynôme est X et dont les racines sont les coordonnées du vecteur v.

On peut stocker un polynôme dans une variable. Le nom de l'indéterminée du polynôme peut être toute chaîne de un à quatre caractères. On peut ensuite effectuer des opérations algébriques grâce aux symboles +, -, * et ∧.

Exemples :

```
--> P=poly([1,-2,0,-1,3],'X','c')
 P  =
          3    4
   1 -2X -X +3X
--> typeof(P)
 ans  =
 polynomial
--> Q=poly([1,-3,2],'X')
 Q  =
          3
   6 -7X +X
--> P+Q
 ans  =
           4
   7 -9X +3X
```

```
--> P*Q
 ans  =
             2    3     4     5   6    7
   6 -19X +14X -5X +23X  -21X  -X  +3X
--> Y=poly(0,"Y");
--> Z=1+2*Y+Y^2
 Z  =
          2
   1 +2Y +Y

--> Z^2
 ans  =
           2    3   4
   1 +4Y +6Y +4Y  +Y
```

Si P est un polynôme et x un vecteur, alors

`degree(P)`	renvoie le degré du polynôme P
`horner(P,x)`	évalue le polynôme P en x (c'est-à-dire calcule P(x)) coordonnée par coordonnée en utilisant l'algorithme d'Hörner
`coeff(P)`	renvoie un vecteur dont les coordonnées sont les coefficients de P
`roots(P)`	renvoie un vecteur dont les coordonnées sont (des approximations de) les racines complexes de P

Exemples :

```
--> X=poly(0,'X');
--> degree(X^4-X^2+2)
 ans  =
   4.
-->P=(1+2*X+X^2)^2; coeff(P)
 ans  =
   1.   4.   6.   4.   1.
horner(P,[1,-1])
 ans  =
   16.   0.
```

```
--> roots(P)
 ans  =
  -1.0001722
  -1. + 0.0001722i
  -1. - 0.0001722i
  -0.9998279
--> Q=poly([%i,2,%i,-1],'Y');roots(P)
 ans  =
   2. - 2.056D-16i
  -1.
   1.735D-08 + 1.i
  -1.735D-08 + 1.i
```

Enfin notons que l'on peut également former des fractions rationnelles en divisant deux polynômes à l'aide du symbole /. Les fonctions `numer` et `denom` permettent d'isoler respectivement le numérateur et le dénominateur d'une fraction rationnelle.

Exemples :

```
--> (X^2+3*X+1)/(X^2-1)
 ans  =
            2
   1 + 3X + X
   -----------
          2
     -1 + X
--> typeof(R)
 ans  =
 rational
```

```
--> horner(R,2)
 ans  =
   3.6666667
--> S=R+X/(X^2+2);
--> numer(S)
 ans  =
           2    3   4
   2 +5X +3X +4X  +X
--> denom(S)
 ans  =
       2   4
  -2 +X  +X
```