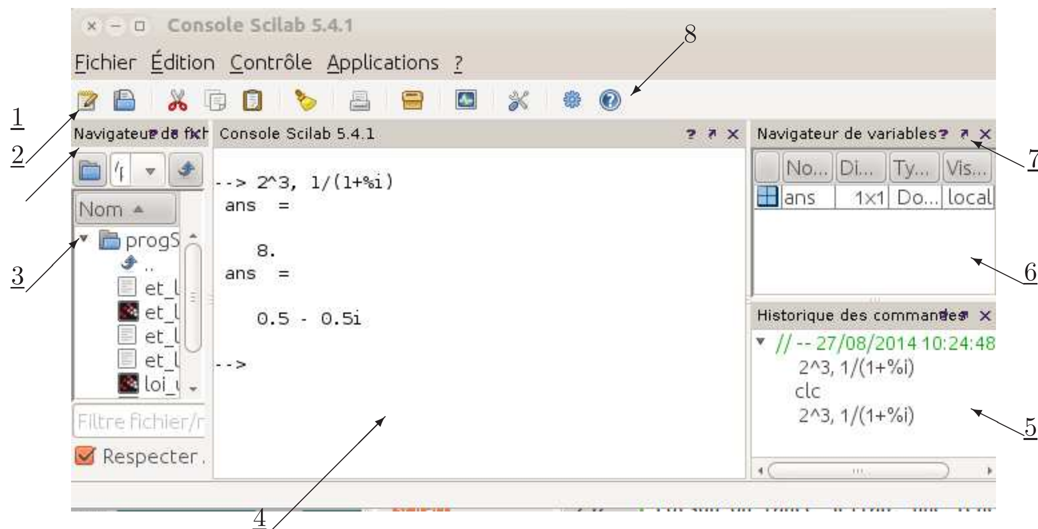
Introduction à Scilab [1]

1 Les caractéristiques principales

— Scilab, abréviation de Scientific laboratory, est un logiciel de calcul numérique libre et open-source (site officiel `http://www.scilab.org`).
— C'est un langage interprété, dont l'élément de base est la matrice.
— Il n'est pas nécessaire de faire des déclarations de variables, ni de préciser la taille des matrices utilisées.

2 L'accès

La version accessible sur les ordinateurs des bâtiments 425 et 440 est la version 5.5. Vous pouvez l'obtenir en allant dans le menu **applications** (en haut à gauche de l'écran), puis dans le sous-menu **Autres**, et en sélectionnant l'icône de Scilab.



1. Menu *Fichier* qui permet entre autres de quitter Scilab proprement.
2. Icône qui ouvre l'éditeur de texte SciNotes dans une fenêtre séparée.
3. Navigateur de fichiers permettant de choisir le répertoire courant de Scilab et d'ouvrir un fichier dans l'éditeur de texte de Scilab.
4. Fenêtre servant à taper les commandes ;
5. Zone donnant l'historique des commandes tapées par l'utilisateur dans la fenêtre de commandes ;
6. Zone où s'affichent les variables créées par l'utilisateur ;
7. Icône qui permet de séparer la fenêtre de navigation des variables du reste : en cliquant dessus, on obtient deux fenêtres indépendantes. Pour attacher une fenêtre séparée dans la fenêtre principale de Scilab, la déplacer jusqu'à la zone désirée, en maintenant le bouton de la souris enfoncé sur la barre contenant le titre de la fenêtre et le '?'.
8. Icône qui ouvre le navigateur d'aide dans une fenêtre séparée.

3 L'environnement

3.1 La fenêtre de commandes

Lorsqu'on lance Scilab, une fenêtre apparaît composée de plusieurs sous-fenêtres (depuis la version 5.4). La sous-fenêtre principale est celle appelée “Console Scilab” ; Le caractère '`-->`' visible dans cette fenêtre signifie que l'ordinateur attend une instruction. Une ligne de commandes que l'on tape dans cette fenêtre est exécutée lorsqu'on appuie sur la touche `Entrée`.

1. Polycopié fait avec la version 5.4.1 de Scilab.
On trouvera les codes de certains exemples dans le répertoire `/commun/doc/lemaire/M322`

Exemple 1.

```
--> 2^3, 1/(1+%i)
 ans  =
    8.
 ans  =
    0.5 - 0.5i
```

```
--> ans*2
 ans  =
    1. - i
```

NB :
- La virgule permet de séparer des instructions écrites sur une même ligne.
- Le nom des constantes prédéfinies commence par un `%`.
- Le résultat d'un calcul est affecté par défaut à une variable appelée `ans` que l'on peut réutiliser dans un autre calcul.
- L'opérateur puissance ^ s'obtient en utilisant la combinaison de touches AltGr+9 ou bien en tapant un accent circonflexe puis un espace. On peut aussi utiliser ** pour désigner la puissance.

`exit` ou (touches Ctr + q)	permet de quitter Scilab
`clc`	efface le contenu de la fenêtre de commandes
touches `Ctr` +`c`	arrête l'exécution d'une commande Scilab.
touches →, ←, ↑↓	permet de se déplacer dans les lignes de commandes tapées dans la fenêtre de commandes.
touche Tab (⇄)	permet la complétion d'une commande à partir de ses premières lettres.
`//`	marque le début d'un commentaire (la suite de la ligne n'est pas exécutée par Scilab).

`diary('`*nomdefichier*`')`	ouvre un fichier (journal de bord) intitulé *nomdefichier* qui gardera la trace de tout ce qui s'affichera dans la fenêtre de commandes.
`diary('`*nomdefichier*`','pause')`	suspend la copie de la fenêtre de commandes dans *nomdefichier*
`diary('`*nomdefichier*`','on')`	reprend la copie de la fenêtre de commandes dans *nomdefichier* si elle a été supendue
`diary(0)`	arrête l'écriture du journal de bord

NB : La fonction `diary` permet de constituer un fichier contenant une copie d'écran du contenu de la fenêtre de commandes. Le fichier obtenu n'est pas un fichier exécutable. Pour écrire une suite de commandes et les sauvegarder comme un fichier exécutable, on utilisera un éditeur de texte (par exemple celui de Scilab) et on donnera l'extension `.sce` au fichier crée (cela sera détaillé page 5).

Exercice 1. Analyser les résultats obtenus en tapant la ligne de commandes suivante :
`cos(%pi/2), x = 1+125e-8, y = x-1; 10*y`

```
 ans  =
    6.123D-17
 x  =
    1.0000012
 ans  =
    0.0000125
```

Les valeurs obtenues sont-elles celles attendues ? Quel est le rôle du point-virgule ? du signe = ?
La ligne de commandes a créé 2 variables `x` et `y`. La valeur affectée à la variable `x` est écrite en notation scientifique, le contenu de la variable `y` est affiché en notation décimale et est arrondi. La valeur affichée de `y` correspond-elle à la valeur que Scilab garde en mémoire ?

NB : On fait suivre une commande par un point-virgule lorsqu'on ne veut pas que le résultat de la commande s'affiche dans la console.
La valeur d'un calcul affichée dans la console est un arrondi et non le contenu exact de la valeur conservée en mémoire. On peut changer le format d'affichage avec la fonction `format`.

Exemple 2.

```
--> format()    // affiche le format actuel
ans  =
    1.    10.   // affichage utilisant 10 caractères en comptant le signe
--> %pi, format(20), %pi, format('e',10), %pi
%pi  =
    3.1415927
 %pi  =
    3.14159265358979312
 %pi  =
    3.142D+00
```

Le format par défaut est `format('v',10)`.

3.2 L'espace de travail

Scilab dispose d'un espace de travail sur l'ordinateur qui garde en mémoire les variables créées et les lignes de commandes exécutées dans la fenêtre de commandes.
Les fichiers créés par Scilab sont sauvegardés dans son répertoire courant. De même, Scilab va chercher, dans ce répertoire courant, les fichiers qu'on lui demande d'ouvrir ou d'exécuter. La commande `pwd` permet de connaitre ce répertoire courant. On peut le changer, soit en utilisant le menu de la console, soit en utilisant la commande linux :
`cd` *adresse_nouveau_repertoire* .

Commande	Description
`pwd`	affiche le nom du répertoire courant pour Scilab
`cd` *rep*	change le répertoire courant pour Scilab qui devient *rep*
`who`	donne la liste des variables présentes dans l'espace de travail
`who_user`	donne uniquement la liste des variables créées par l'utilisateur et les fonctions qu'il a appelées
`whos`	donne la liste des variables présentes dans l'espace de travail ainsi que leurs propriétés (type et taille)
`exists`('*var*')	retourne 1 si une variable de nom *var* existe et retourne 0 sinon
`clear` $var1$ $var2$	efface les variables $var1$ et $var2$ de l'espace de travail
`clear`	efface toutes les variables créées dans l'espace de travail
`save(`*'nom_fichier.sod'*, 'var_1', ..., 'var_n'`)`	sauve la valeur des variables var_1, ..., var_n dans un fichier binaire *nom_fichier.sod*
`load(`*'nom_fichier.sod'*`)`	permet de récupérer toutes les données sauvegardées dans le fichier *nom_fichier*.

NB : L'extension `sod` signifie *Scilab Open Data*. Le format des fichiers `.sod` permet de conserver à la fois le nom et le contenu des variables sauvegardées. Ce ne sont pas des fichiers imprimables.

Exemple 3.

```
--> clear

-->a = 2; b=3; c = a*b;

-->who_user
 Vos variables sont :
 c         b         a         home
 Utilise 19 éléments sur 4990752
 -->save('sauvegarde.sod','a','c')

-->clear

-->a
  !--error 4
Variable non définie: a

-->load('sauvegarde.sod')

-->a
 a  =
    2.

-->c
 c  =
    6.
 -->exists('b')
 ans  =
    0.
```

NB : `home` est une variable de type « chaîne de caractères » contenant l'adresse du répertoire de travail de Scilab.

NB : En tapant la commande `who`, on voit apparaître :
- les variables qui ont été créées depuis l'ouverture de Scilab ;
- les noms des fonctions utilisées ;
- les noms des bibliothèques de Scilab (qui se terminent par lib) (les fonctions et les bibliothèques sont considérées comme des variables par Scilab).
- les constantes prédéfinies dont le nom commence par `%` comme π, e, l'unité imaginaire i, la précision machine `eps`, et les deux autres constantes classiques de l'arithmétique flottante (`nan` pour '*not a number*') et `inf` (pour ∞) ;
- le nombre de mots de huit octets (64 bits) utilisés, ainsi que la mémoire totale disponible (taille de la pile), puis le nombre de variables utilisées ainsi que le nombre maximum autorisé.

Nom des variables Le nom d'une variable ne doit pas commencer par un chiffre. Il doit être constitué d'au plus 24 caractères chacun pouvant être une lettre (sans accent), un chiffre ou le caractère souligné (underscore en anglais). Scilab fait la distinction entre minuscules et majuscules.

Codage des nombres réels Scilab utilise la représentation des nombres réels en *virgule flottante* en suivant la norme IEEE-754 et travaille en double précision, ce qui signifie que les réels sont stockés en mémoire sur 64 bits : un nombre réel non nul est approché par un nombre ayant une représentation finie en base 2 que l'on écrit $(-1)^s 2^e (1 + \sum_{j=1}^{t} a_j 2^{-j})$ avec

- $s \in \{0, 1\}$ suivant le signe du réel ($s = 0$ pour un réel positif) ; il occupe 1 bit,
- $m = a_1 \ldots a_t$ la mantisse ; elle occupe 52 bits ($a_1 = 1$ n'est pas codé),
- e entier entre -1022 à 1023, appelé l'exposant ; $e + 1023$ dispose de 11 bits.

L'exposant $e = -1023$ est réservé pour représenter 0. L'exposant $e = 1024$ sert à représenter les nombres spéciaux $\pm$inf et NaN (not a number). Un tel codage permet de conserver des réels entre environ 2.10^{-308} et 2.10^{308} avec 16 chiffres significatifs.

Exemple 4. Le tableau présente des exemples de codage de nombres en virgule flottante normalisée :

Nombre	codage de l'exposant +1023	codage de la mantisse
2^{-1022}	000 0000 0001	que des bits 0
$(2 - 2^{-52})2^{1023}$	111 1111 1110	que des bits 1
30.0625	100 0000 0011	1110 0001 suivi de 43 zéros

Par exemple, 30.0625 s'écrit en binaire 11 110.0001. L'exposant est donc 4 et le codage en binaire de 1027 est 100 0000 0011.

Les opérations élémentaires $op \in \{+, -, \times, /\}$ entre 2 réels x et y se font alors de la façon suivante : $\mathrm{fl}(\mathrm{fl}(x) \text{ op } \mathrm{fl}(y))$.

Exercice 2.

1. Quels sont les nombres qui ont un développement binaire fini ?
2. Pour quels a, le calcul $(1 + a) - 1$ donne 0 avec un logiciel qui utilise la norme IEEE-754 ?

Les exercices suivants présentent quelques conséquences de ce codage :

Exercice 3. Analyser le résultat des commandes suivantes :

```
--> sqrt(2)^2 - 2,  x  = 1e28,  y = 1e10, x + y - x
 ans  =
    4.441D-16
 x  =
    1.000D+28
 y  =
    1.000D+10
 ans  =
    0.
```

Quelle est l'erreur relative dans le calcul suivant ?

```
  -->  a = 1e-15, b = (1 + a) - 1
 a  =
    1.000D-15
 b  =
    1.110D-15
```

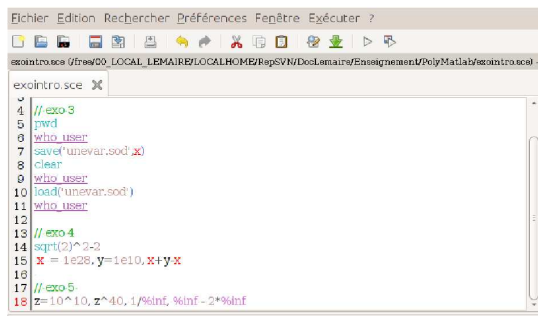
Exercice 4. Analyser le résultat des commandes suivantes :

```
--> z = 10^10, z^40, 1/%inf,  %inf - 2*%inf
 z  =
    1.000D+10
 ans  =
    Inf
 ans  =
    0.
 ans  =
    Nan
```

NB : La constante `%inf` se comporte comme l'infini ∞ en mathématique.

3.3 L'éditeur de texte SciNotes

Scilab dispose d'un éditeur de texte appelé SciNotes. On l'ouvre en cliquant sur la première icône (représentant un bloc note et un crayon) dans la barre de menu de la console de Scilab. Plutot que d'écrire les commandes les unes après les autres dans la console, on peut les écrire dans l'éditeur SciNotes (de préférence une par ligne pour une meilleure visibilité). On peut alors exécuter une commande ou un bloc de commandes en sélectionnant les commandes à exécuter avec la souris, puis en tapant en même temps sur les touches Ctrl et E (ou en faisant un clic droit pour sélectionner "Exécuter la sélection"). On peut Les lignes de commandes sélectionnées seront exécutées l'une après l'autre. Pour sauvegarder le fichier, lui donner un nom suivi de l'extension `.sce` afin qu'il soit reconnu par Scilab comme un fichier exécutable.



Quelques raccourcis clavier de l'éditeur SciNotes

Raccourci	Action
Ctrl S	sauvegarde le fichier
Ctrl Z	annule la dernière modification
Ctrl E	exécute le fichier jusqu'à la position du curseur avec écho (i.e. en affichant les commandes exécutées et les réponses pour les commandes qui ne sont pas terminées par un ;)
Ctrl L	exécute tout le fichier avec écho

Exercice 5.

1. Trouver quel est le répertoire courant pour Scilab.
2. Créer un répertoire *TPscilab* en utilisant soit le gestionnaire de fichiers sous Ubuntu, soit la commande linux `mkdir TPscilab` à partir d'un terminal. Changer le répertoire courant de Scilab pour que ce soit le sous-répertoire *TPscilab*.
3. Ouvrir l'éditeur SciNotes.
4. Aller dans la sous-fenêtre de Scilab contenant l'historique des commandes. afin de faire un copié-collé des lignes de commandes que vous avez tapé jusqu'à maintenant (pour sélectionner un ensemble de lignes dans l'historique, cliquer sur la 1ère ligne à copier, taper sur la touche 'Majuscule' tout en cliquant sur la dernière ligne à copier. On peut ensuite déplacer avec la souris le bloc de lignes sélectionnée dans la fenêtre de SciNotes ou utiliser le le clic droit de la souris pour faire un copié-collé.)
5. Sauvegarder le fichier en lui donnant le nom `tpintro.sce`. Essayer les différentes possibilités de l'éditeur pour exécuter tout le fichier, pour exécuter seulement un bloc de commandes.

Le fichier `tpintro.sce` pourra être complété en tapant les commandes proposées dans la suite du polycopié. Pour ajouter des lignes de commentaire, il suffit de précéder chaque ligne de commentaires par `//`.

3.4 L'aide en ligne

Pour trouver les fonctions prédéfinies dans Scilab afin d'effectuer une tache donnée ou pour savoir comment utiliser une commande Scilab, le bon réflexe est de consulter l'aide en ligne de Scilab.

On accède à l'aide en ligne :
- depuis la barre de menu en cliquant sur ? puis sur 'Aide de Scilab'
- depuis la barre d'outils en cliquant sur l'icône en forme de ?
- depuis la console en tapant l'une des commandes suivantes :

Commande	Description
`help`	donne la liste de toutes les commandes par thèmes
`help` *nom*	ouvre la fenêtre d'aide sur le descriptif de la commande *nom*
`apropos` *nom*	permet de rechercher une commande à partir du mot clé *nom*; On obtient la liste des commandes en rapport avec le mot *nom*.

Exercice 6. Taper la commande `apropos arrondi` afin de trouver quatre fonctions qui permettent d'arrondir un réel. Les tester sur les réels 1.4, 1.5, 1.6, -1.4 et -1.5 par exemple. Noter les différences entre ces fonctions.

NB : On peut exécuter les exemples donnés dans l'aide en cliquant sur le triangle dessiné dans l'encadré des exemples. On peut aussi sélectionner des lignes de commandes d'un exemple et faire un clic droit avec la souris pour copier la sélection dans l'éditeur (la sélection est copiée dans un nouveau fichier).

4 Les types de données de base

Une variable appelée *nom* est créée au moment où une valeur lui est affectée à l'aide du signe '='. Elle possède un type qui dépend de son contenu et qui peut changer dynamiquement.

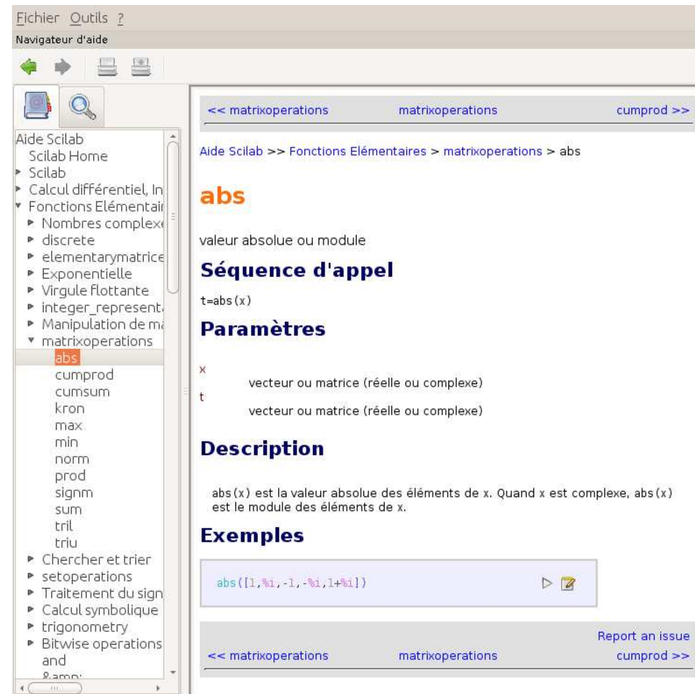


FIGURE 1 – Fenêtre obtenue en tapant la commande `help abs`. L'exemple proposé peut être exécuté en cliquant sur le triangle.

4.1 Les principaux types scalaires

Les données usuelles telles que les booléens, les nombres réels/complexes, les chaînes de caractères, les polynômes et les fractions rationnelles ont des types qui les distinguent.

Donnée scalaire	%t, %f	nb réel ou complexe	polynome	fraction rationnelle	chaîne de caractères
Type	`boolean`	`constant`	`polynomial`	`rational`	`string`

Une même opération sur des objets de type différents ne signifiera pas la même chose. D'autres part, à chaque type correspond des fonctions particulières pour les manipuler.

Exemple 5. `2.3 + 4` donne le nombre flottant 6.3.
`'aa'+'bb'` donne la chaîne de caractères 'aabb'.

4.2 Le type matrice

Les données énumérées dans le tableau précédent sont en fait traitées comme des matrices de taille 1×1. La matrice est l'objet de base dans Scilab. Une matrice est un tableau dont les coefficients sont tous de même type, l'un de ceux énumérés dans le tableau. Les vecteurs sont vus comme des matrices à une ligne ou une colonne.

4.3 Création d'une matrice

Les matrices peuvent être construites en entrant explicitement la liste des éléments : on entre les éléments d'une ligne en les séparant par un blanc ou une virgule. Pour séparer les éléments de deux lignes, on met un point-virgule.

Exemple 6. Voici trois exemples de matrices, avec le type et la taille de chacune :

```
--> A = [ 4, 5, 6 ; 1, 3, 2 ; 9, 8, 7 ], typeof(A), size(A)
 A  =
    4.    5.    6.
    1.    3.    2.
    9.    8.    7.
 ans  =
 constant
 ans  =
    3.    3.
```

```
--> B = [1,%s^2+1,%s+1], typeof(B), size(B), length(B)
 B  =
               2
   1     1 + s     1 + s
 ans  =
 polynomial
 ans  =
   1.    3.
 ans  =
   3.
```

```
--> C = [ 'le', 'la', 'un' ; 'du', 'de','des'  ], typeof(C), size(C)
C  =
!le  la  un   !
!             !
!du  de  des  !
ans  =
 string
ans  =
    2.    3.
```

On peut aussi définir une matrice en utilisant des fonctions prédéfinies de Scilab.

`linspace`(x, y, n)	définit un vecteur ligne constitué de n nombres espacés régulièrement entre x et y
`zeros`(n, m)	crée une matrice de taille $n \times m$ dont tous les éléments sont nuls
`ones`(n, m)	crée une matrice de taille $n \times m$ dont tous les éléments valent 1
`eye`(n, m)	crée une matrice de taille $n \times m$ avec des 1 sur la diagonale et des zéros ailleurs.
`matrix`(v, n, m)	transforme un vecteur ou une matrice v ayant nm coefficients en une matrice de taille $n \times m$ en remplissant les colonnes les unes après les autres avec les éléments de v
`repmat`(A, n, m)	crée une matrice en répétant la matrice A n fois sur chaque ligne et m fois sur chaque colonne
`diag`(v)	si v est un vecteur, crée une matrice diagonale dont les coefficients diagonaux sont ceux de v

Cas particulier : l'instruction `deb:pas:fin` définit un vecteur `x=[x(1),...,x(n)]` avec `n` le plus grand entier tel que `x(n)`$\leq$ `fin` et `x(i)=deb+(i-1)*pas` pour tout i (à comparer avec `linspace`). Si le paramètre `pas` n'est pas précisé, il vaut 1.

Exemple 7.

```
--> E = [1 2 3; 4 5 6]
 E  =
   1.    2.    3.
   4.    5.    6.

--> EE = matrix(E,3,2)
 EE  =
   1.    5.
   4.    3.
   2.    6.
```

```
--> F = 3:6
F  =
    3.    4.    5.    6.

--> L = 1 : -0.15 : 0
 L  =
    1.    0.85    0.7    0.55    0.4    0.25
      0.1
```

On voit sur cet exemple qu'il y a un ordre implicite sur les coefficients d'une matrice, les coefficients sont parcourus colonne par colonne.

NB : Attention, certaines fonctions peuvent avoir un nombre variable de paramètres. Par exemple, si M désigne une matrice, `zeros`(M) définit une matrice de même taille que M n'ayant que des zéros.

Exercice 7.

1. Que donne la commande `ones(7)` ?
2. Quelle est la commande permettant de définir une variable appelée x contenant la liste des entiers impairs inférieurs à 50 ?

4.4 Extraction de valeurs

Si A est une matrice, $A(i, j)$ renvoie à l'élément de la i-ème ligne et de la j-ème colonne d'une matrice A. Pour un vecteur v, il suffit d'écrire $v(i)$ pour avoir le i-ème élément, que le vecteur soit en ligne ou en colonne.

NB :

- i et j sont nécessairement des entiers strictement positifs.
- $ peut être utilisé pour indiquer l'indice du dernier élément d'une ligne ou d'une colonne ($-1 correspond alors à l'indice de l'avant-dernier élément ...)

On peut aussi extraire une sous-matrice d'une matrice :

Exemple 8.
- `A(2,:)` permet d'extraire la 2ème ligne de la matrice `A`.
- `A([2,3],$)` permet d'extraire le vecteur colonne constitué des 2-ième et 3-ième éléments de la dernière colonne de `A`.

NB : `A(:)` retourne un vecteur colonne contenant tous les éléments de `A` parcourus colonne après colonne.

Exemple 9. Avec le vecteur F défini à l'exemple 7, les commandes `F($-2), A = matrix(F,2,2)` donnent les résultats suivants

```
ans  =
   4.
A=
   3.   5.
   4.   6.
```

4.5 Modification d'une matrice

Avec Scilab, il n'y a pas besoin de déclarer les types des variables utilisées, ni de spécifier leur taille. On peut donc changer la taille d'une variable au cours d'un programme :
- [ ] désigne la matrice vide ;
- Il est possible de concaténer des matrices en utilisant [ ] pour en faire une plus grande, si bien sûr les dimensions des matrices sont cohérentes.

Exemple 10. Avec la matrice B définie dans l'exemple 6 :

```
--> B2 = [B; ones(1,3)]
 B2  =
                  2
    1     1 + s     1 + s

    1     1         1

--> B2(3,2) = %s
```

```
B2  =
              2
   1     1 + s     1 + s

   1     1         1

   0     s         0
```

NB : On voit sur le dernier exemple, que l'élément (3,2) de la matrice `B2` n'existant pas, Scilab a transformé `B2` en une matrice 3×3 en complétant par des 0.

Exercice 8. Quelles sont les commandes permettant de
1. définir w comme le vecteur colonne de longueur 3 ne contenant que des 1 ?
2. définir la matrice $M = \begin{pmatrix} 2 & 3 \\ 4 & 0 \\ 1 & 5 \end{pmatrix}$?
3. créer une matrice R de taille 3×3 en ajoutant à M le vecteur colonne w ?

NB : Lorsque cela est possible, il est recommandé de fixer la taille des matrices que l'on va utiliser au début d'un programme : par exemple, la commande `M=zeros(n,m)` définit `M` comme une matrice $n \times m$ dont les éléments sont tous 0. Un espace mémoire est ainsi alloué à `M`.

5 Les opérations sur les matrices

5.1 Les opérations matricielles

Les opérations matricielles telles que, $+$ l'addition, $-$ la soustraction, $*$ la multiplication, ^ la puissance, $\backslash$ la division à gauche et / la division à droite, s'obtiennent avec la même syntaxe que pour les opérations sur les scalaires. La transposition d'une matrice A s'écrit $A.'$

NB : A' est la transposée de la conjuguée de la matrice A.
Si les dimensions des matrices A et B sont compatibles, $A\backslash B$ (resp. B/A) donne la solution X de $AX = B$ (resp. $XA = B$) éventuellement au sens des moindres carrés [2].

Scilab contient beaucoup de fonctions pour le calcul matriciel telles que `trace, det, spec, kernel, lu, qr, svd ...` (voir la section « Algèbre linéaire » dans l'aide en ligne).

2. Une équation de la forme $Ax = b$ n'a pas toujours de solution, mais il est possible de déterminer x de sorte de minimiser $||Ax - b||_2 = \sum_i (\sum_j A_{i,j} x_j - b_i)^2$; c'est ce qu'on appelle une solution au sens des moindres carrés.

Exercice 9. 1. Définir un vecteur colonne ayant 3 éléments égaux à 1 et définir R comme la matrice $R = \begin{pmatrix} 2 & 3 & 1 \\ 4 & 0 & 1 \\ 1 & 5 & 1 \end{pmatrix}$.

2. Déterminer l'image du vecteur w par la matrice R.
3. Déterminer la valeur x_1 de la solution de l'équation $R.x = w$.
4. On pose $w_2 = w + \delta w$ avec $\delta w = \begin{pmatrix} 0.01 \\ -0.01 \\ 0.01 \end{pmatrix}$. Déterminer la valeur x_2 de la solution de l'équation $R.x = w_2$.

   Comparer les valeurs de $\frac{||x_2 - x_1||}{||x_1||}$ et de $\frac{||\delta w||}{||w||}$ (utiliser l'aide de Scilab pour trouver une fonction qui calcule la norme d'un vecteur). Commenter en précisant la norme choisie.

5.2 Les opérations sur chaque élément d'une matrice

En faisant précéder d'un point les opérateurs `*`, `\`, `/` et `^`, on réalise des opérations élément par élément : `.\`, `./` et `.^` peuvent donc s'utiliser entre 2 matrices de même taille, mais aussi entre un scalaire b et une matrice A (le scalaire b est vu comme `b*ones(A)`).

Exemple 11.

```
--> [ 4, 5, 6 ; 1, 3, 2].^2
ans =
    16.    25.    36.
     1.     9.     4.
```

NB : Attention, 1. est considéré comme l'entier 1 d'où des confusions : par exemple, pour obtenir la matrice dont le coefficient (i, j) est $2/A(i, j)$ pour tout i, j, on peut écrire $(2)./A$ ou $2 \quad ./A$ mais pas $2./A$.

Exemple 12.

```
--> A=[2,3;1,2]; B=1./A, C=(1)./A
 B  =
    2.  - 3.
  - 1.    2.
 C  =
    0.5    0.3333333
    1.     0.5
```

La plupart des fonctions scalaires de Scilab sont faites pour travailler aussi bien sur un scalaire que sur tous les éléments d'une matrice. Il faut en profiter, cela évite d'écrire des boucles et cela permet une exécution plus rapide.

Exemple 13.

```
--> X = [0, 1; 0.5, 0.25];
--> cos(%pi*X)
 ans  =
    1.          - 1.
    6.123D-17     0.7071068
```

NB : Si X est une matrice $n \times m$ de réels, l'instruction `Y = exp(X)` crée une matrice $n \times m$ telle que $Y(i, j) = \exp(X(i, j))$ pour tout i, j. Pour obtenir l'exponentielle d'une matrice carrée Z définie par mathématiquement par la somme $\sum_{k=0}^{+\infty} \frac{1}{k!} Z^k$, on dispose de la fonction `expm`.

5.3 Quelques fonctions opérant sur une dimension de la matrice

Certaines fonctions opérant sur les coefficients d'une matrice, peuvent aussi opérer séparément sur chaque colonne ou sur chaque ligne d'une matrice en ajoutant un paramètre. Par exemple, si `nom_fonction` désigne une des fonctions du tableau suivant et M est une matrice, on a 3 syntaxes possibles :

— `nom_fonction(M)` : la fonction `nom_fonction` opère sur la matrice M ;
— `nom_fonction(M,'c')` pour que la fonction `nom_fonction` opère sur chaque ligne et retourne donc un vecteur colonne ;
— `nom_fonction(M,'r')`, la fonction `nom_fonction` opère sur chaque colonne et retourne donc un vecteur ligne.

`min`(M)	retourne la valeur minimale des coefficients de M
`max`(M)	retourne la valeur maximale des coefficients de M
`sum`(M)	retourne la somme des éléments de M
`cumsum`(M)	retourne une matrice de même taille que M contenant la somme cumulée des premiers éléments de M.
`mean`(M)	retourne la moyenne de tous les éléments de M
`prod`(M)	retourne le produit des éléments de M
`cumprod`(M)	retourne un vecteur contenant le produit cumulé des premiers éléments de M.
`gsort`(M)	tri par ordre décroissant des éléments de M

Exemple 14. Avec la matrice $M = \begin{pmatrix} 2. & 3. \\ 4. & 0. \\ 1. & 5. \end{pmatrix}$, la commande `cumsum(M,'r')` donne la matrice $\begin{pmatrix} 2. & 3. \\ 6. & 3. \\ 7. & 8. \end{pmatrix}$.

Exercice 10. On se propose dans cet exercice de calculer une valeur approchée de e^{-20} de 2 façons différentes et de les comparer.

1. On pose $n = 65$ et $a = -20$. Définir un vecteur u de taille n tel que $u(i) = \frac{a}{i}$ pour tout $i \in \{1, \ldots, n\}$. en utilisant les opérations élément par élément sur les vecteurs.
2. Construire, à l'aide de u, un vecteur v de même taille que u tel que $v(i) = \frac{a^i}{i!}$ pour tout i.
3. Calculer, en utilisant le vecteur v, la valeur $s_1 = \sum_{i=0}^{n} \frac{a^i}{i!}$. Comparer $s1$ à e^a.
4. Déterminer de la même façon la valeur s_2 de $\sum_{i=0}^{n} \frac{20^i}{i!}$. Comparer $1/s2$ à e^{-20}.
5. Refaire les mêmes calculs avec $n = 100$ puis $n = 150$. Commenter les résultats obtenus.

***Exercice 11.** `gsort` permet aussi d'ordonner par ordre croissant et de récupérer la permutation utilisée pour ordonner les coefficients, regarder l'aide pour voir comment. Faire afficher la permutation qui permet d'ordonner par ordre croissant les éléments de la première colonne de M.

6 Opérateurs relationnels et logiques

Le tableau suivant décrit les différents opérateurs relationnels dont le résultat sera un booléen (F ou T) et les opérations booléennes classiques :

Opérateurs relationnels	<,	<=,	>=,	== (égalité),	~= (différent)
Opérateurs logiques	& (et),	| (ou),	~ (non)		

NB : Bien faire la différence entre "==" et "=" qui sert à affecter une valeur à une variable.

Utilisés avec des matrices, les opérateurs relationnels et logiques énumérés dans le tableau précédent, sont évalués sur chaque élément de la matrice.

Exemple 15. `y = (x > 0) & (x < 5)` est une matrice de booléens de mêmes dimensions que x telle que

$$y(i,j) = \begin{cases} T & \text{si } x(i,j) > 0 \text{ et } x(i,j) < 5 \\ F & \text{sinon.} \end{cases}$$

Exemple 16.

```
- -> x = [0 -1 2 -10 3];
- -> y = (x > 1)
y =
     F F T F T
```

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

NB : On peut utiliser les opérateurs logiques avec des réels. Dans ce cas, 0 est identifié au booléen F et tout réel non nul est associé au booléen T.

Exemple 17. Avec le vecteur x défini dans l'exemple précédent :

```
--> y = 2*(x > 1)
 y  =
    0.    0.    2.    0.    2.
```

NB : Dans l'exemple, `x>1` est de type 'boolean' alors que `y` définit une variable de type 'constant'.
Beaucoup de fonctions qui acceptent comme paramètres des matrices de réels, acceptent aussi des matrices de type boolean (T étant vu comme 1 et F comme 0), mais pas toutes.

Ordres de priorité par ordre décroissant d'importance : opérations arithmétiques puis opérations relationnelles puis opérations logiques.

Exemple 18.

```
--> 2*3<4
ans =
     F
```

```
--> 2*(3<4)
ans =
     2.
```

Exercice 12. Comment définir un vecteur ligne `x` contenant 100 valeurs équiréparties entre -5 et 5 et un vecteur `y` de même taille que `x` et telle que

$$y(i) = \begin{cases} \sin(1/x(i)) \text{ pour } i \text{ tel que } |x(i)| > 0.1 \\ 0 \text{ sinon} \end{cases}$$

en seulement 2 commandes ?

6.1 Représentations de points dans le plan

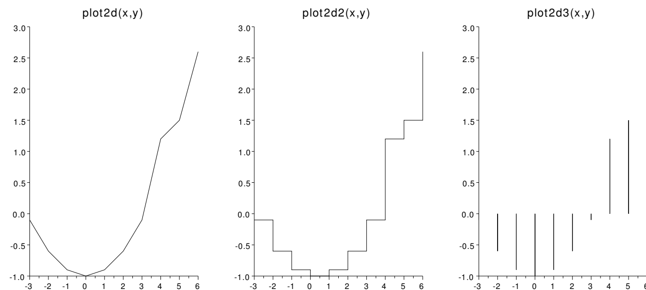
Il existe plusieurs possibilités pour représenter un ensemble de points $(x(i), y(i))$. Les plus utilisées sont énumérées dans le tableau :

`plot2d(x,y,opt_arg)`	les points sont reliés par des segments de droites
`plot2d2(x,y,opt_arg)`	les points sont reliés par un segment horizontal suivi d'un segment vertical
`plot2d3(x,y,opt_arg)`	diagramme en bâtons

Exemple 19. Les figures suivantes donnent différentes représentations des points suivants :

x	-3	-2	-1	0	1
y	-0.1	-0.6	-0.9	-1.0	-0.9

x	2	3	4	5	6
y	-0.6	-0.1	1.2	1.5	2.6



`opt_arg` désigne une liste optionnelle de propriétés qui s'écrivent sous la forme `nom_option = valeur` et sont séparées par des virgules. Les principales sont :

Nom de l'option	Valeurs possibles	Descriptif
style	entier k	si $k \in \{1, \ldots, 32\}$, k désigne le numéro de la couleur du tracé. si $k \in \{-14, \ldots, 0\}$, les points ne sont pas reliés, le symbole identifié par $-k$ est utilisé pour marquer la position des points
rec	[xmin, ymin, xmax, ymax]	décrit la zone du graphique visible
frameflag	entier $k \in \{0, \ldots, 9\}$	pour gérer les échelles en x et y

NB : Le tracé d'une fonction f qui n'est pas affine par morceaux se fait donc de façon approchée : pour obtenir un tracé sur un intervalle borné $[a, b]$, on définit un vecteur $x = [x_1, \ldots, x_n]$ formé d'une discrétisation de l'intervalle $[a, b]$: $x_1 = a < x_1 < \ldots < \ldots < x_{n-1} < x_n = b$, on définit un vecteur $y = [y_1, \ldots, y_n]$ contenant les valeurs de f en chaque point x_i : $y_i = f(x_i)$ pour tout i et on utilise la commande `plot2d(x,y)` pour tracer la ligne polygonale passant par les points de coordonnées (x_i, y_i), $1 \leq i \leq n$.

NB : Les différentes valeurs des options sont décrites dans l'aide de `plot2d` (voir aussi l'exemple 19). La commande `color('color_name')` retourne le le numéro de la couleur dont le nom est color_name.
Les commandes `getcolor()` et `getmark()` affichent un tableau permettant d'avoir la liste des couleurs et des marqueurs pour chaque numéro.

Exemple 20. Les 3 lignes de commandes suivantes permettent d'obtenir un tracé approché de la fonction cosinus en rouge sur l'intervalle $[0, 2\pi]$:

```
x = 0:0.1:2*%pi;
y = cos(x);
plot2d(x, y, style = color('red')}
```

Exercice 13. (suite de l'exercice 12). Tracer la fonction f définie par $f(x) = \begin{cases} \sin(1/x) \text{ si } |x| > 0.1 \\ 0 \text{ sinon} \end{cases}$ sur $[-5, 5]$.

6.2 La superposition de tracés

Par défaut, les tracés effectués par différentes instructions de type `plot2d` sont superposés et l'échelle est recalculée à chaque nouveau tracé qui s'ajoute pour s'ajuster aux minima et aux maxima de toutes les courbes de sorte que toutes soient visibles (i.e. `frameflag=9`). La commande `clf()` efface le contenu de la fenêtre graphique active.

Exemple 21. Les commandes suivantes permettent d'avoir, sur une même figure une courbe noire et une courbe bleue représentant respectivement les fonctions cosinus et sinus sur l'intervalle $[0, 2\pi]$

```
x = 0:0.1:2*%pi;
y1 = cos(x); y2 = sin(x) ;
plot2d(x, y1, style=1)
plot2d(x, y2, style=2)
```

6.3 Gestion de la fenêtre graphique et ajout de légendes

Le tableau suivant décrit quelques commandes pour gérer l'affichage de figures dans une fenêtre graphique :

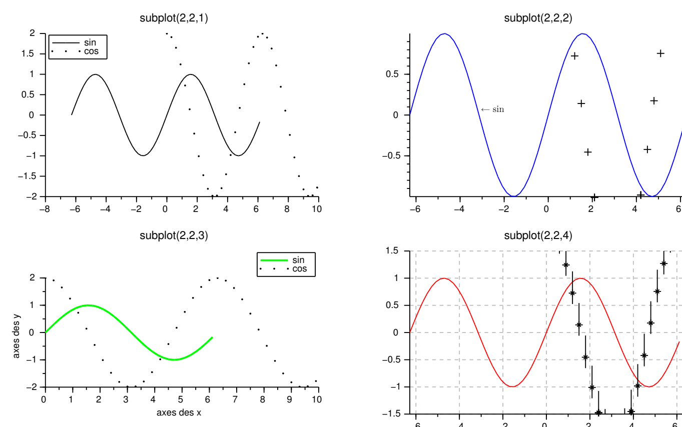
`clf`	efface le contenu de la fenêtre graphique active (en ouvre une si aucune fenêtre graphique n'existe)
`clf(num)`	efface le contenu de la fenêtre graphique de numéro `num`
`scf(num)`	s'il n'y a pas de fenêtre de numéro `num`, la créée et retourne le pointeur associé à cette fenêtre. Si cette fenêtre existe, elle la rend active
`xdel` (resp. `xdel(num)`)	ferme la fenêtre graphique active, (resp. celle de numéro `num`).
`subplot`(n, m, p)	partage la fenêtre graphique en $m \times n$ espaces graphiques et sélectionne le p-ième.
`drawlater()`	met en attente les affichages
`drawnow()`	met à jour le contenu de la fenêtre graphique.

Le tableau suivant décrit quelques commandes pour gérer les échelles d'une figure et ajouter du texte sur une figure :

`replot([xmin, ymin, xmax, ymax])`	pour modifier la partie de la figure affichée
`square(xmin,ymin,xmax,ymax)`	pour obtenir une échelle isométrique de la zone précisée
`xgrid(k)`	quadrillage du graphique ; k est le numéro de la couleur du quadrillage
`title`('titre')	titre pour le graphique
`xlabel`('titre')	légende pour l'axe des x
`ylabel`('titre')	légende pour l'axe des y
`legend`('titre1','titre2',...)	légende pour chaque courbe du graphique, (on peut préciser sa position en ajoutant un paramètre optionnel)
`xstring`$(x, y,$'texte')	texte à la position (x, y)

Exemple 22. [3]

Pour illustrer l'utilisation des différentes commandes présentées dans les deux tableaux précédentes, voici une figure et ci-dessous le programme qui permet de créer cette figure.




```
clear;clf()
x1 = -2*%pi:0.2:2*%pi; y1 = sin(x1);
x2 = -0:0.3:10; y2 = 2*cos(x2);
drawlater()
subplot(2,2,1)
plot2d(x1,y1,style=1) // tracé en lignes
                      // brisées noires
plot2d(x2,y2,style=0) // tracé des points
                      // avec le marqueur .
legend('sin','cos',2) // legende positionnée
                      // en haut à gauche
title('subplot(2,2,1)')

subplot(2,2,2)
plot2d(x1,y1,style=2) // tracé en  bleu
// superposition avec l'echelle définie par le 1er
    graphique
plot2d(x2,y2,style=-1,frameflag=7)
```

3. l'adresse du fichier contenant ce programme est `/commun/doc/lemaire/M322/exsuperpopgraph.sce`

```
// affichage d'une formule latex dans la fenêtre
xstring(-%pi,0,'$ \leftarrow \sin$')
title('subplot(2,2,2)')

subplot(2,2,3)
plot2d(x1,y1,style=3) // tracé en vert
// repère orthonormé
plot2d(x2,y2,style=0,frameflag=4)
title('subplot(2,2,3)')
xlabel('axes des x'); ylabel('axes des y')
legend('sin','cos') // legende en haut à droite (par
    défaut)

subplot(2,2,4)
plot2d(x1,y1,style=5) // tracé en rouge
plot2d(x2,y2,rect=[-2*%pi,-1.5,2*%pi,1.5],..
style=-10,frameflag=7)
errbar(x2,y2,0.2,0.4) // ajout de barres d'erreur
title('subplot(2,2,4)')
xgrid(color("gray")) // ajout d'une grille
                     //de couleur numero 33
drawnow()
```

7 Les fonctions et programmes

7.1 Les fonctions

La plupart des commandes de Scilab utilisées jusqu'à maintenant sont des fonctions. La syntaxe d'une fonction est la suivante :

```
function [res1, ..., resn]= nomdelafonction (var1, ..., vars)

     // lignes d'instructions définissant les variables res1, ...., resn
     // à l'aide des paramètres d'entrée var1, ..., vars

endfunction
```

- `nomdelafonction` doit être composé d'au plus 24 caractères, uniquement des lettres non accentuées ou des chiffres ou des caractères _ (underscore) et ne doit pas commencer par un chiffre.
- `var1`,..., `vars` sont les arguments d'appel de la fonction
- `res1`,..., `resn` sont les arguments de sortie

On écrit une telle fonction dans un fichier texte dont le nom a pour extension `.sci`

On peut faire afficher le contenu d'une fonction en utilisant la commande `edit('nomdelafonction')`.

Exemple 23. La commande `edit('sec')` ouvre le fichier sec.sci dans SciNotes dont le contenu est le suivant :

```
// Scilab ( http://www.scilab.org/ ) - This file is part of Scilab
// Copyright (C) INRIA, Serge Steer
//
// This file must be used under the terms of the CeCILL.
// This source file is licensed as described in the file COPYING, which
// you should have received as part of this distribution.  The terms
// are also available at
// http://www.cecill.info/licences/Licence_CeCILL_V2-en.txt
function y = sec(x)
// Secant
  y = ones(x)./cos(x);
endfunction
```

NB : Attention à ne pas modifier les fichiers de Scilab.

Utilisation. Une fois que l'on a écrit une fonction appelée `nomdelafonction` dans un fichier et et qu'on l'a sauvegardée dans le répertoire de travail de Scilab sous le nom `nomdelafonction.sci` , il faut charger la fonction dans l'environnement de Scilab pour qu'elle soit reconnue. Pour cela, on tape la commande

`exec`('nomdelafonction.sci')

ou on utilise le menu *Exécuter* ou les raccourcis dans la barre d'outils de SciNotes. Les fonctions que l'on crée se rajoutent aux fonctions de Scilab. Elles s'utilisent de la même façon.

NB : Ne pas oublier de recharger la fonction après chaque modification de sa définition.

NB : Si le fichier `nomdelafonction.sci` se trouve dans un autre répertoire que le répertoire de travail de Scilab, on peut le charger en utilisant la commande `exec('`*adresse_rep/nomdelafonction.sci*`')`.

Variables locales et arguments de la fonction Les variables utilisées dans le corps d'une fonction autres que les arguments de sortie sont des variables internes. Les modifications apportées à un argument d'entrée dans le corps de la fonction ne sont pas visibles à l'extérieur de la fonction, sauf si l'argument d'entrée est aussi un argument de sortie de la fonction.

Exemple 24. Analyser ce que fait la fonction suivante[4] et quelles sont les variables internes.

```
function [m, e, a, b] = etstat(x, p)
// x et p sont des vecteurs de même taille, les valeurs de p sont supposées strictement positives.
    s = sum(p) ;  m = sum(x.*p)/s ;
    a = min(x);   b = max(x);
    e= b-a;
endfunction
```

Une fois que l'on a enregistré le fichier `etstat.sci` dans le répertoire de travail de Scilab, les lignes de commandes suivantes donnent un exemple d'utilisation de cette fonction. Analyser ce qu'elles font.

```
--> exec('etstat.sci')
--> r = etstat([1,5,3,4],[1,2,2,1])
--> [u, v] = etstat([1,5,3,4],[1,2,2,1])
->r, u, v
 r  =
    3.5
 u  =
    3.5
 v  =
    4.
```

NB : Il est commode de ranger les variables de sortie `res1`,..., `resn` d'une fonction par ordre décroissant d'intérêt car pour avoir accès à la valeur de la variable `res2` par exemple, il faut appeler la fonction avec au moins deux variables de sortie ; si on tape la commande

`[a1,a2]=nomdelafonction(b1,..., bs)`

on aura dans `a1` (resp. `a2`) la valeur de `res1` (resp. `res2`) calculée à partir des paramètres d'entrée `b1`, . . . , `bs`.

NB : Attention, si, dans la définition d'une fonction, est utilisée une variable qui n'est pas un argument d'entrée de la fonction et n'est pas définie dans le corps de la fonction, Scilab donnera comme valeur à cette variable la valeur d'une variable de même nom si elle existe dans l'espace de travail ou affichera un message d'erreur.

Exemple 25. Dans la fonction `ajout` ci-dessous, `a` sera considéré comme une variable globale. Si elle n'existe pas dans l'espace de travail de Scilab, son exécution donnera un message d'erreur.

```
 function y = ajout(x)
   y = x + a
 endfunction
```

7.2 Les programmes

Un programme, appelé aussi *script*, est un fichier texte dont le nom a pour extension `.sce` et qui est constitué simplement d'une suite de commandes Scilab. C'est souvent une succession d'appels à des fonctions, chacune des fonctions étant chargée de faire un calcul particulier effectué à partir des valeurs des paramètres d'entrée. En conséquence, un programme contiendra au début autant de lignes de commande de type

`exec('adresse_de_fonctions_pers')`

qu'il y a de fonctions utilisées dans le programme ne faisant pas partie des fonctions de Scilab. Le tableau suivant présente quelques commandes permettant d'agir sur le déroulement d'un programme :

4. le fichier `etstat.sci` se trouve dans le répertoire /commun/doc/lemaire/M322

`disp`(*var*)	affiche le contenu de *var* dans la fenêtre de commande
`disp`(*varn*,...,*var*1)	affiche le contenu de *var*1, *var*2, ..., *varn* dans la fenêtre de commande sur des lignes différentes.
rep=`input`(*'texte'*)	affiche la chaîne de caractères *texte* et donne la main à l'utilisateur pour qu'il tape une matrice qui sera affectée à la variable *rep*.
`pause`	arrête l'exécution d'un programme : un symbole d'invite de commande est affiché indiquant le "niveau" de la pause (e.g. `-1->`). L'utilisateur peut alors faire afficher le contenu des variables du niveau correspondant, reprendre l'exécution en tapant "return", arrêter l'exécution en tapant "abort".
`sleep`(n)	arrête l'exécution d'un programme pendant n millisecondes
`halt`('message')	affiche le texte 'message' et arrête l'exécution d'un programme jusqu'à ce qu'on tape sur la touche 'Entrée' du clavier.

NB : Si une ligne d'instructions est trop longue, on peut la couper en terminant la ligne coupée par ...

On peut convertir une matrice quelconque en une matrice de chaînes de caractères avec la commande `string`. Cela sera utile pour afficher le contenu de variables dans une fenêtre graphique.

Exemple 26. Le fichier tirage.sce[5] contient les lignes de commandes suivantes :

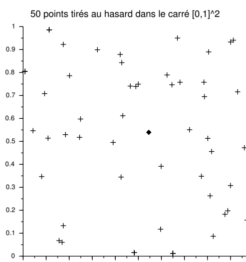
```
clear
clf()
n = input('Combien de points voulez-vous tirer ? ');
X = rand(2,n);   // matrice contenant des réels choisis au hasard entre 0 et 1
m = mean(X,'c'); // calcule, pour chaque ligne de X,  la valeur moyenne des éléments
plot2d(X(1,:), X(2,:), style = -1, frameflag = 4)
title(string(n) + ' points tirés au hasard dans le carré [0,1]^2')
disp(m, 'Le centre de gravite du nuage de points est :');
plot2d(m(1,1), m(2,1), style = -4)
```

Voici un exemple d'exécution du programme et la figure obtenue :

```
--> exec('tirage.sce',-1)
Combien de points voulez-vous tirer ? 50

 Le centre de gravite du nuage de points est :

    0.5451245
    0.5396453
```



Exercice 14. (*Approximation d'une dérivée*) Soit f une fonction dérivable définie sur un intervalle $[a,b]$. Supposons que l'on connaisse la valeur de f en les points de la subdivision de $[a,b]$ de pas constant $h=(b-a)/n$. On pose $x_i = a+ih$ et $f_i = f(x_i)$ pour tout $i \in \{0,\ldots,n\}$. On définit le schéma de différences finies à gauche par $\delta_g(i) = \frac{1}{h}(f_i - f_{i-1})$, le schéma de différences finies à droite par $\delta_d(i) = \frac{1}{h}(f_{i+1} - f_i)$ et le schéma de différences finies centrées par $\delta_c(i) = \frac{1}{2h}(f_{i+1} - f_{i-1})$.

1. *Etude du schéma de différences finies à gauche.*
   (a) Du point de vue géométrique, utiliser $\delta_g(i)$ revient à approcher la dérivée de f au point x_i par la pente d'une droite. Laquelle ? (faire un dessin).
   (b) Ecrire une fonction `deriveedisc.sci` qui :
      — a 2 arguments d'appel, un vecteur y et un réel positif h, de sorte que les valeurs de y correspondent aux valeurs d'une fonction f en les points d'une subdivision de $[a,b]$ de pas constant h ;
      — a un argument de sortie : le vecteur dg contenant les valeurs du schéma de différences finies à gauche pour le vecteur y et le pas de discrétisation h.
   (c) Ecrire une fonction `sinabs.sci` qui calcule les valeurs de la fonction $g : t \mapsto \sin(|t|^{3/2})$ et de sa dérivée en tous les points d'un vecteur x (elle aura un argument d'appel et 2 arguments de sortie).

5. Le fichier se trouve dans le répertoire /commun/doc/lemaire/M322.

(d) Ecrire un programme `etderivdisc.sce` qui :
- crée une subdivision régulière de l'intervalle $[a, b] = [-3, 3]$;
- divise la fenêtre graphique en 2 sous-figures ;
- représente sur une des sous-figures, les courbes de la fonction g et de sa dérivée g' ;
- calcule le schéma de différences finies à gauche pour la fonction g et le pas $h = 0.01$;
- représente sur une 2ème sous-figure, la courbe de la différence entre g' et le schéma de différences finies à gauche.
- détermine quel est l'écart maximal entre g' et le schéma de différences finies à gauche sur l'intervalle $[1, 2]$ et sur l'intervalle $[-1, 1]$ et donne, à chaque fois, un point où cet écart est maximal (voir l'aide de la fonction `max` pour cela).

(e) Lancer le programme. Commenter les résultats obtenus.

2* *Etude des deux autres schémas.*

(a) Du point de vue géométrique, utiliser $\delta_d(i)$ ou $\delta_c(i)$ revient à approcher la dérivée de f au point x_i par la pente des droites. Lesquelles ? (faire un dessin).

(b) Compléter la fonction `deriveedisc.sci` pour qu'elle retourne, comme second argument de sortie, le vecteur dc contenant les valeurs du schéma de différences centrées pour le vecteur y et le pas de discrétisation h.

(c) Modifier le programme fait précédemment pour :
- qu'il divise la fenêtre graphique en 3 sous-figures,
- qu'il représente aussi sur la 2ème sous-figure, la courbe de la différence entre g' et le schéma de différences finies à droite pour le même pas de discrétisation h ;
- qu'il représente sur la 3ème sous-figure, la courbe de la différence entre g' et le schéma de différences finies centrées pour le même pas de discrétisation h ;
- qu'il détermine quel est l'écart maximal entre g' et chaque schéma sur l'intervalle $[1, 2]$ et sur l'intervalle $[-1, 1]$ et donne, à chaque fois, un point où cet écart est maximal.

(d) Lancer le programme avec $h = 0.01$ puis $h = 10^{-4}$. Commenter les résultats obtenus.

8 Les commandes structurées

8.1 La boucle `for`

On utilise une boucle `for` pour exécuter des lignes de commandes un nombre prédéfini de fois.

`for` *variable = v*
    instructions
`end`

v désigne en général un vecteur ligne : chaque élément de v est affecté, l'un après l'autre à la variable '*variable*' et pour chacune de ces valeurs, les *'instructions'* sont exécutées.

NB : v peut aussi être une matrice. Dans ce cas *variable* recevra le contenu des colonnes de v les unes après les autres.

Exemple 27. Ces quelques lignes calculent $n!$ pour $n = 1$ à 100.

```
n = 100; fact = 1;
for k = 2:n
  fact = fact*k;
end
```

NB : On peut aussi taper simplement :

```
n = 100;
fact = prod(1:n);
```

L'exécution du premier programme prend plus de temps que l'exécution du deuxième programme. Il faut privilégier les opérations vectorielles à l'utilisation de boucles dans Scilab.

Exercice 15. La fonction suivante permet de construire la matrice de Vandermonde $A = \begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^n \\ \vdots & \vdots & \vdots & \vdots & \\ 1 & x_m & x_m^2 & \dots & x_m^n \end{pmatrix}$.

```
function A=vandm(x,n)
 // x est un vecteur ligne
 // n est un entier positif
  m = length(x)
  A = zeros(m,n+1)
  for i = 1:m
     for j = 1:n+1
       A(i,j) = x(i)^(j-1)
     end
  end
endfunction
```

1. Utiliser la fonction `timer()` pour déterminer le temps d'exécution de cette fonction pour $n = 1000$ et x un vecteur définissant une discrétisation de l'intervalle $[0, 1]$ en 1000 points.
2. Taper `//` devant la ligne `A = zeros(m,n+1)` pour qu'elle ne soit pas exécuter, recharger la fonction vandm.sci dans Scilab et déterminer de nouveau le temps de calcul de la matrice de Vandermonde pour les mêmes valeurs pour voir l'intérêt d'initialiser la matrice A.
3. Faites une copie du fichier vandm.sci en vandm2.sci et modifier le contenu de vandm2 pour faire le calcul de la matrice de Vandermonde en utilisant une seule boucle `for`. Y-a-t-il un gain en temps de calcul par rapport à la fonction vandm de départ ? Si oui de combien ?

***Exercice 16.** Les polynômes de Chebyshev de 1ère espèce sont définis par la relation de récurrence :

$$f_{n+2}(X) = 2X\ f_{n+1}(X) - f_n(X) \text{ avec } f_1(X) = 1 \text{ et } f_2(X) = X.$$

Ecrire une fonction qui calcule les k premiers polynomes de Chebyshev en des points donnés dans un vecteur. Le resultat sera écrit sous forme d'une matrice, la i-ème colonne contenant les valeurs calculées pour le i-ème polynôme.
Pour tester cette fonction, tracer les 5 premiers polynomes sur l'intervalle $[-1, 1]$ sur une même figure et comparer la courbe de f_5 avec celle de $x \mapsto \cos(4\text{Arccos}(x))$.

8.2 L'instruction conditionnelle `if`

`if` *conditions* `then`
 instructions
`end`

Les '*instructions*' ne sont exécutées que si les '*conditions*' sont vérifiées, plus précisément si '*conditions*' a une valeur différente de 0.

NB : le mot-clé `then` est facultatif, il doit être écrit sur la même ligne que `if`.

Une variante possible est :

`if` *conditions* `then`
 instructions (exécutées si les '*conditions*' sont vérifiées)
`else`
 instructions (exécutées si les '*conditions*' ne sont pas vérifiées)
`end`

Plusieurs instructions `if` peuvent être emboitées, par exemple :

`if` *conditions_1* `then`
 instructions_1 (exécutées si les '*conditions_1*' sont vérifiées)
`elseif` *conditions_2* `then`
 instructions_2 (exécutées si les '*conditions_1*' ne sont pas vérifiées et si les '*conditions_2*' sont vérifiées)
⋮
`elseif` *conditions_k* `then`
 instructions_k (exécutées si les '*conditions_1*', ..., '*conditions_(k-1)*' ne sont pas vérifiées et si les '*conditions_k*' sont vérifiées)
`else` (facultatif)
 instructions (exécutées si aucune des '*conditions_1*',..., '*conditions_k*' ne sont vérifiées)
`end`

Exemple 28. Considérons la fonction f définie par $f(x) = \begin{cases} -(x-1)^2 \text{ si } x \leq 1 \\ \ln(x) \text{ si } 1 < x \leq 4 \\ \ln(4) + \exp(-x) \text{ si } x > 4. \end{cases}$

Pour déterminer la valeur de f en plusieurs points, on peut se calquer sur la description mathématique de f :

```
function y=valeurf1(x)
// x est un vecteur ligne
y = zeros(x)
for k = 1:length(x)
   xx = x(k)
   if xx >= 4 then
      y(k) = log(4) - exp(-xx)
   elseif xx >= 1 then
      y(k) = log(xx)
   else
      y(k) = (xx-1)^2
   end
end
endfunction
```

On peut optimiser la fonction `valeurf1.sci` en supprimant la boucle `for` grâce la fonction `find` de Scilab qui permet de faire les opérations sur les vecteurs au lieu de les faire point par point :

```
function y=valeurf2(x)
  // x est un vecteur ligne
  y = zeros(x)
  I1 = find(x <= 1)
  I2 = find(x > 1 & x <= 4)
  I3 = find(x > 4)
  y(I1) = (x(I1)-1).^2
  y(I2) = log(x(I2))
  y(I3) = log(4) - exp(-x(I3))
endfunction
```

NB : Une proposition logique (par exemple, `(x>0 & x<10)`) a pour valeur `T` si elle est vraie et `F` sinon. Cette valeur est en général transformée par 1 et 0 respectivement si nécessaire. Dans les calculs, on peut quelque fois éviter d'utiliser l'instruction `if` en introduisant des indicatrices d'ensembles.

8.3 La boucle conditionnelle `while`

La boucle conditionnelle « tant que » est utilisée pour exécuter une série de commandes tant qu'une expression logique est satisfaite.

`while` *conditions*
    instructions
`end`

Les '*instructions*' sont exécutées tant que les '*conditions*' sont vérifiées.

Exemple 29. L'exemple ci-dessous calcule par dichotomie le plus petit nombre qui ajouté à 1 donne un résultat numérique différent de 1 (ce nombre est la précision relative de la machine).

```
x = 1;
while 1+x > 1
  x = x/2;
end
disp(x)
```

Instructions de rupture/prolongation	
`break`	termine l'exécution de la boucle la plus proche dans laquelle se trouve `break`
`continue`	passe directement à l'instruction suivante de la boucle courante
Touches Ctr + c	interrompt l'exécution du programme, l'exécution du programme peut reprendre en tapant `resume`, peut être arrêté en tapant `abort`
`error`('*message*')	affiche la chaîne de caractères *message* et interrompt l'exécution du programme en cours

Exercice 17. La pseudo-inverse d'une fonction croissante H à valeurs dans $[0,1]$ est définie sur $]0,1[$ par

$$H^{-1}(u) = \inf(t \in \mathbb{R}, H(t) \geq u).$$

Soit G la fonction définie sur $\mathbb{R}$ par $G(t) = 0$ si $t < 1$ et $G(t) = \sum_{k=1}^{[t]} \frac{6}{\pi^2 k^2}$ si $t \geq 1$.
Ecrire une fonction `invsom.sci` qui détermine la valeur de G^{-1} en un réel x si $x \in]0,1[$ et qui affiche un message d'erreur sinon (on utilisera une boucle `while`).

8.4 La programmation récursive

La programmation récursive consiste à appeler une fonction à l'intérieur d'elle-même.

Exemple 30. Voici deux façons de programmer la fonction factorielle, la première est dite itérative, la seconde est dite récursive.

```
function y=fact1(n)
  y = 1;
  for k = 2:n
    y = y*k;
  end
 endfunction
```

```
function y = fact2(n)
  if n <=1 then
     y = 1;
  else
     y = n*fact2(n-1);
  end
 endfunction
```

Dans l'algorithme récursif, il y a une condition d'arrêt : la fonction `fact2`(n) ne retourne un résultat que si $n \leq 1$, sinon le résultat est mis en attente puisqu'il faut, avant cela, que la fonction `fact2(n-1)` retourne son résultat.

C'est une programmation qui peut être couteuse en place mémoire.

Exercice 18. Soit n un entier positif et a un réel. L'algorithme suivant propose de calculer a^n de manière récursive. Ecrire une fonction `puissance.sci` basée sur cet algorithme récursif.

```
Si n=0
   retourner 1
sinon si n est pair
         calculer p=a^(n/2) et retourner p*p*a
      sinon
         calculer p=a^((n-1)/2) et retourner p*p
```

Indications : on pourra utiliser la fonction `pmodulo`.

***Exercice 19.** Une approximation du triangle de Sierpiǹski s'obtient en utilisant l'algorithme récursif suivant :

1. Commencer à partir d'un triangle équilatérale ayant une base parallèle à l'axe des abscisses.
2. Enlever le triangle central dont les sommets sont les milieux des côtés du triangle précédent.
3. Recommencer à la deuxième étape avec chacun des petits triangles restants.

En utilisant la fonction `xfpoly` qui permet dessiner un polygone dont l'intérieur est colorié, écrire une fonction qui dessine une approximation du triangle de Sierpiǹski après un nombre fixé d'itérations (attention, à bien écrire la condition d'arrêt).
Ecrire ensuite un programme qui

— demande à l'utilisateur combien d'itérations, il veut avec pour limite 7 itérations,
— définit n comme le minimum entre l'entier entrée par l'utilisateur et 7,
— détermine les coordonnées des sommets du 1er triangle,
— exécute la commande `drawlater()` qui met en attente le tracé [6], puis lance la fonction pour n itérations,
— définit une échelle isométrique à l'aide de la commande `square` et enfin exécute la fonction `drawnow()`.

Tester ce programme avec $n = 2$, puis $n = 4$ et enfin $n = 7$ itérations (ne pas aller au dela de 7 itérations pour éviter de bloquer Scilab)

9 Passer une fonction comme argument d'une autre fonction

La plupart des fonctions de Scilab et toutes les fonctions que l'on créé sont des variables de type function. Elles peuvent être mises comme paramètre d'une autre fonction.

Le tableau suivant donne des exemples de fonctions Scilab dont un des paramètres d'entrée est une fonction.

6. Sans la commande `drawlater()`, Scilab modifie la figure à chaque commande de tracé ce qui ralentit énormément le tracé d'une figure complexe.

`intg`	évaluation numérique d'une intégrale simple
`int2d, int3d`	évaluation numérique d'une intégrale double et d'une intégrale triple respectivement
`fsolve`	résolution numérique de la solution d'un système d'équations
`fminsearch`	recherche du minimum d'une fonction.
`ode`	solveur d'équations différentielles ordinaires

Exemple 31. Pour déterminer numériquement l'intégrale de 1 à 2 de la fonction $x \mapsto x^2$, on écrira :

```
function y=carre(x), y=x.^2, endfunction;
intg(1,2,carre)
```

On peut aussi passer une fonction avec des paramètres comme argument de ces fonctions. On utilise alors la commande `list` qui permet de créer une liste d'éléments :

Exemple 32. Les commandes suivantes font la même chose que dans l'exemple précédent :

```
function y=puissance(x,a), y=x.^a, endfunction;
intg(1,2,list(puissance,2))
```

L'exemple ci-dessous montre comment on peut programmer une fonction qui appelle une autre fonction en paramètre, il n'y a rien de particulier si la fonction n'a pas de paramètre :

Exemple 33.

```
function y=derivdisc(x,h,f)
 // calcule la dérivée discrète de f en  chacun des points du vecteur x avec le pas h,
 // en utilisant le schéma de différences finies à droite,
 // h peut etre un réel strictement positif ou un vecteur de réels strictement positifs
 //   de même taille que x
 y=(f(x+h)-f(x))./h;
 endfunction
```

Pour calculer ensuite la dérivée discrète de la fonction sinus au point 2 avec un pas de 0.01, on écrira : `derivdisc(2,0.01,sin)`.

Il est plus compliqué de programmer une fonction qui admet comme argument une fonction ayant un nombre indéterminé de paramètres comme c'est le cas de la fonction `intg`.

Exemple 34. Voici une façon simple de changer la fonction `derivdisc` de l'exemple précédent afin de pouvoir l'appliquer avec une fonction ayant au plus un paramètre.

```
function y=derivdisc(x,pas,f,a)
 // calcule la dérivée discrète de f en  chacun des points du vecteur x avec le pas h,
 // en utilisant le schéma de différences finies à droite,
 // h peut etre un réel strictement positif ou un vecteur de réels strictement positifs
 //   de même taille que x
 // f doit etre une fonction définie avec au plus un paramètre dont l'appel
 //   se fait avec f(x,a)
 if argn(2)<3  then error('entrer au moins 3 paramètres')
 elseif argn(2)==3 then
     y=(f(x+pas)-f(x))./pas;
 else
     y=(f(x+pas,a)-f(x,a))./pas;
 end
endfunction
```

NB : `argn` est une fonction qui retourne un vecteur composé du nombre d'arguments de sortie de la fonction et du nombre de paramètres d'entrée.

Exercice 20. Evaluer l'intégrale $\int_0^{100} \exp(-x^2)dx$ à l'aide de la fonction `intg`. Comparer le résultat obtenu avec ce que donne la fonction `erf`.

A Complément sur la manipulation des booléens

A.1 Les fonctions `and` et `or`

Les fonctions `and` et `or` sont des opérateurs logiques qui s'appliquent globalement à un matrice booléenne :

`or`(M)	retourne le booléen T si au moins un des éléments de M vaut T et retourne le booléen F sinon.
`and`(M)	retourne T si tous les éléments du vecteur x ont la valeur T et retourne le booléen F sinon.

`and` et `or` peuvent aussi s'utiliser pour tester une condition sur chaque ligne ou chaque colonne d'une matrice en rajoutant le paramètre 'c' ou 'r' respectivement.

Exemple 35.

```
--> D=[1,2,3;-2,1,4];

--> or(D > 3)
 ans  =
  T
```

```
--> and(D > 0, 'c')
 ans  =
  T
  F
```

A.2 La fonction `find`

La fonction `find` permet de savoir quels sont les indices des élements d'une matrice qui satisfont à une proposition logique.

I =`find`(M)	retourne la liste des numéros des éléments égaux à `T` de la matrice M.
[I, J]=`find`(M)	retourne les indices des lignes (dans le vecteur I) et des colonnes (dans le vecteur J) des éléments égaux à `T` de la matrice M.

NB : Si M est une matrice de réels, `find(`M`)` correspond à `find(`$M \sim= 0$`)` puisque tout élément non nul est traité comme le booléen `T`.

Exemple 36. Avec la matrice D de l'exemple précédent,

```
--> [x,y] = find(D>2)
 y  =
    3.    3.
 x  =
    1.    2.
```

```
--> l = find(D>2)
 l  =
    5.    6.
```

B Conversion d'objets

`bool2str(A)`	convertit une matrice booléenne A en une matrice de 0 et de 1 de même taille
`string(A)`	convertit une matrice A en une matrice de chaînes de caractères de même taille que A.
`double(A)`	convertit une matrice A constituée d'entiers à 1,2 ou 4 octets en matrice de flottants double précision

C L'analyse du fonctionnement d'une fonction

C.1 Temps d'exécution d'un programme

On dispose de plusieurs fonctions pour déterminer le temps d'exécution d'un programme :

`tic()`	lance un chronomètre
`toc()`	affiche le temps écoulé (en secondes) depuis le dernier `tic`
`timer()`	retourne le temps CPU (i.e. le nombres de cycles processeur) depuis le dernier appel à `timer()`

C.2 Le débogage d'une fonction

Que faire si une fonction ou un programme ne fonctionne pas ?

- Si l'exécution s'arrête avant la fin du programme, un message d'erreur s'affiche, il contient des informations sur le type d'erreur commise.
- Par défaut, lors de l'exécution d'une fonction, le résultat d'une affectation ne s'affiche pas même si elle n'est pas terminée par un point-virgule. Pour faire un affichage, on peut utiliser `disp` pour afficher le contenu de certaines variables. On peut aussi changer le mode d'exécution en commençant la fonction par `mode(0)` pour faire afficher les résultats des commandes qui ne sont pas suivies de ;
- On peut aussi ajouter la commande `pause` à des endroits bien choisis de la fonction qui arrêtera l'exécution de la fonction et permettra d'avoir la main pour analyser le contenu de variables. La commande `whereami` permet de savoir à quelle ligne, l'exécution s'est arrêtée. L'exécution de la fonction reprend en tapant `return`.

D Complément sur la création de figure

D.1 La superposition de tracés sur une même figure

Avec une seule fonction de type `plot2d(x,y)`, on peut faire différents tracés à condition que les tracés soient constitués du même nombre de points :

- Si tous les tracés utilisent la même liste d'abscisses, x sera encore un vecteur ligne. Par contre, y sera une matrice : les ordonnées des points pour un tracé sont mis dans une colonne de y.
- x et y peuvent aussi être des matrices de même dimension : le i-ième tracé sera effectué avec les points dont les coordonnées sont données par les i-ièmes colonnes de x et y.

Une couleur différente est attribuée par défaut à chaque courbe. On peut aussi définir le style de chaque tracé, la propriété `style` est alors un vecteur.

Exemple 37. Les commandes suivantes permettent de tracer sur une même figure les courbes des fonctions cosinus et sinus à partir d'une discrétisation de l'intervalle $[0, 2\pi]$ définie par la variable `x`. Des couleurs différentes sont automatiquement attribuées à chaque courbe.

```
x = 0:0.1:2*%pi;
y1 = cos(x); y2 = sin(x) ;
plot2d(x,[y1',y2'])
```

D.2 La sauvegarde d'une figure

Une figure peut être sauvegardée sous un format propre à Scilab avec l'extension `.scg`. Pour cela, cliquer sur le bouton `Enregistrer` du menu `Fichier` de la fenêtre graphique et entrer un nom de fichier avec l'extension `.scg` dans l'encadré qui apparaît. Un tel fichier peut être visualisé et modifié en utilisant le bouton `Charger` du menu `Fichier` de la fenêtre graphique.

Une figure peut aussi être exportée sous différents formats d'images en utilisant le menu « Export vers » ou « Export vectoriel vers ».

D.3 Les propriétés d'une figure

Les propriétés d'une fenêtre graphique sont définies par un objet *Figure* qui est une structure hierarchique. Le schéma ci-dessous représente une structure pour une fenêtre divisée en 2 sous-fenêtres :

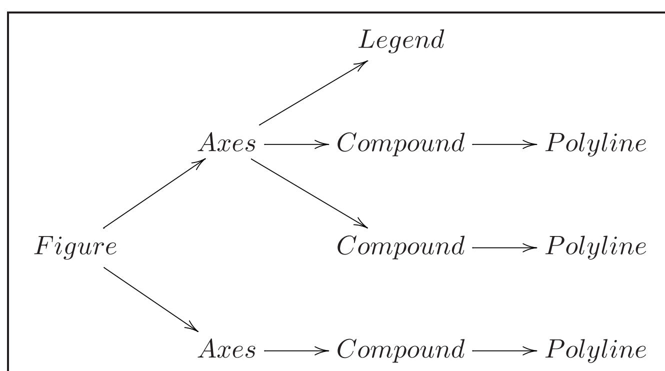


Schéma représentant une structure associée à une fenêtre graphique divisée en deux sous-fenêtres, l'une contenant deux tracés effectués chacun par une commande de type `plot2d` et une légende, l'autre contenant un seul tracé effectué par une commande de type `plot2d`.

On peut accéder à la description de cette structure et modifier les valeurs des propriétes des objets qui la composent par le menu « Edition » en cliquant sur le sous-menu « Propriétés de la figure ».
La suite de cette section explique comment modifier les propriétés des objets d'une figure dans un programme.

Chaque objet a un identificateur (appelé *handle*) qui permet de modifier les propriétés de cet objet et les objets-fils dans l'arbre décrivant la structure. Le tableau décrit les principales commandes pour récupérer l'indentificateur d'un objet de la fenêtre courante :

gcf()	retourne l'identificateur associé à la fenêtre active (abréviation de « *get current figure* »)
gca()	retourne l'identificateur associé aux axes de la fenêtre courante (abréviation de « *get current axes* »)
gce()	retourne l'identificateur associé au dernier tracé effectué (abréviation de « *get current entity* »)

Exemple 38.

Les lignes de commandes ci-contre vont permettre d'expliquer comment modifier certaines propriétés d'une figure (il est recommandé de taper les commandes et de cliquer sur « Propriétés de la figure » du menu « Edition » pour avoir une description de la structure associée à la figure).

```
clf()
x = 0:0.1:2*%pi;
y1 = cos(x); y2 = sin(x);
plot2d(x,y1,style=2)
plot2d(x,y2,style=5)
it = gce();
title('Courbes des sinus et cosinus')
legend('cos','sin');
il = gce();
ia = gca();
```

On récupère dans la variable `it` l'identifiant de l'objet de type *compound* crée par la commande `plot2d(x,y2,style=5)` et dans `il`, l'objet de type *legend* crée par la commande `legend('cos','sin')`. En tapant `it` dans la fenêtre de commandes, on obtient la description ci-contre :

```
--> it
 ans =
Handle of type "Compound" with properties:
==========================================
parent: Axes
children: "Polyline"
visible = "on"
user_data = []
tag =
```

Ici, l'objet n'a qu'un objet fils de type *Polyline*. L'identifiant de cet objet fils est `it.children` dont les propriétés sont décrites ci-contre :
Par exemple, `line_style = 1` signifie que le tracé est une ligne continue. Pour avoir à la place un tracé en pointillés, il suffit de taper la commande `it.children.line_style=2;`
La variable `ia` contient l'identificateur de l'objet *Axes*. L'objet *Axes* est notamment composé d'objets gérant le titre et chacun des axes et contient des objets fils ici trois : deux de type *compound* et un de type *legend*. Par exemple la commande `ia.title.font_size=2` permet d'augmenter la taille des caractères du titre.

```
-->it.children
 ans  =
Handle of type "Polyline" with properties:
==========================================
parent: Compound
children: []
visible = "on"
data = matrix 63x2
closed = "off"
line_mode = "on"
fill_mode = "off"
```

```
fill_mode = "off"
line_style = 1
thickness = 1
arrow_size_factor = 1
polyline_style = 1
foreground = 5
background = -2
interp_color_vector = []
interp_color_mode = "off"
mark_mode = "off"
mark_style = 0
mark_size_unit = "tabulated"
mark_size = 0
mark_foreground = -1
mark_background = -2
x_shift = []
y_shift = []
z_shift = []
bar_width = 0
clip_state = "clipgrf"
clip_box = []
user_data = []
tag =
```

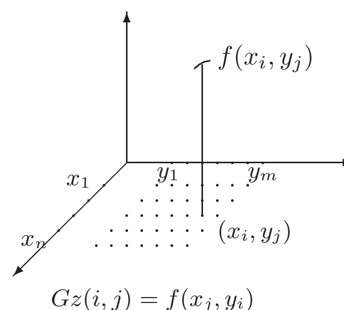
NB : On trouve la liste des propriétés associées aux différents objets graphiques dans l'aide en entrant par exemple le mot-clé *properties*.

D.4 Quelques fonctions pour la représentation graphique en 3D

Pour représenter une fonction $(x, y) \mapsto f(x, y)$ sur un rectangle $I \times J$, on commence par définir des vecteurs $x = [x_1, \dots, x_n]$ et $y = [y_1, \dots, y_m]$ contenant les points d'une discrétisation des intervalles I et J.

La fonction `[Gx,Gy]=meshgrid(x,y)` définit deux matrices de taille $m \times n$:

$$Gx = \begin{pmatrix} x_1 & x_2 \dots & x_n \\ \vdots & \vdots & \vdots \\ x_1 & x_2 \dots & x_n \end{pmatrix} \text{ et } Gy = \begin{pmatrix} y_1 & y_1 \dots & y_1 \\ \vdots & \vdots & \vdots \\ y_m & y_m \dots & y_m \end{pmatrix}.$$



$Gz(i, j) = f(x_j, y_i)$

Gx et Gy contiennent respectivement les abscisses et les ordonnées des points de la grille de $I \times J$ définie par les vecteurs x et y.
On calcule ensuite la fonction f en chaque point de cette grille :

$$Gz(i, j) = f(Gx(i, j), Gy(i, j)) \text{ pour tout } (i, j) \in \{1, \dots, m\} \times \{1, \dots, n\}.$$

On peut alors utiliser l'une des fonctions de représentation d'une surface décrites dans le tableau :

`plot3d(x,y,Gz,arg_opt)`	tracé d'une surface en 3D.
`plot3d1(x,y,Gz,arg_opt)`	tracé d'une surface en 3D.
`Sgrayplot(x,y,Gz,arg_opt)`	représentation 2D d'une surface en projection
`contour(x,y,Gz,L,arg_opt)`	représentation des lignes de niveau d'une surface en projection (L est soit un entier donnant le nb de lignes de niveau, soit un vecteur contenant les valeurs des lignes de niveau

`meshgrid(x,y)`	création d'une grille de points
`colorbar(m,M)`	affichage de la table de couleurs : m et M étant les valeurs minimales et maximales de la fonction tracée
`xset('colormap',nom_colomap)`	pour changer la palette de couleurs utilisée

NB : `arg_opt` est une liste d'options qui s'écrivent sous la forme nom_option=valeur ; les plus importantes sont les angles (en degrés) `alpha` et `theta` qui précisent l'angle de vue de la caméra en coordonnées sphériques (si O est le centre de la boite et Oc la direction de la caméra, alors `alpha` donne l'angle $(Oz; Oc)$ et `theta` donne l'angle entre Ox et la projection de Oc sur le plan Oxy).

Exemple 39. Le programme suivant permet d'obtenir la surface dessinée en haut à gauche sur la figure 2.

```
clear;clf();
x = [-3:0.2:3];
y = [-3:0.2:3];
[Gx,Gy] = meshgrid(x,y);
Gz = exp(-2/3*(Gx.^2+Gy.^2-Gx.*Gy))/(sqrt(3)*%pi);
plot3d(x, y, Gz, alpha=89.5, theta=40)
xtitle('plot3d(x,y,Gz,alpha=89.5,theta=40)')
```

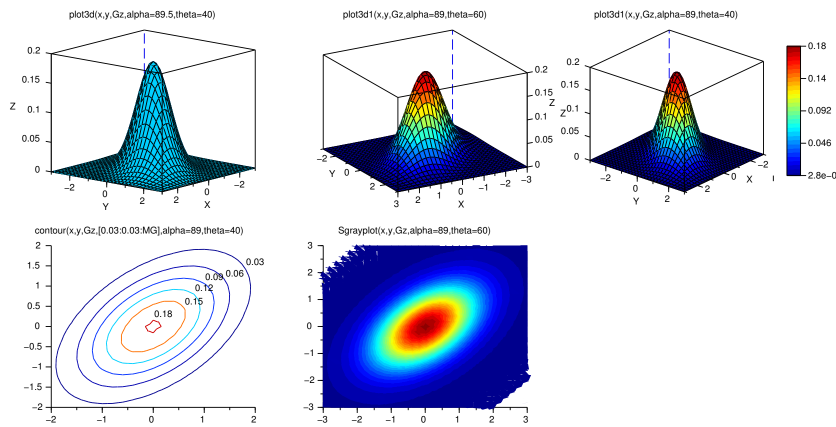


FIGURE 2 – Différents types de représentations de la fonction $(x, y) \mapsto \frac{1}{\pi\sqrt{3}} e^{-\frac{2}{3}(x^2+y^2-xy)}$ (la palette de couleurs utilisée est appelée `jetcolormap`)

E Les chaînes de caractères

E.1 La manipulation des chaînes de caractères

`length`(M)	renvoie une matrice contenant les tailles des chaînes constituant la matrice M.
ch1 `+` ch2	concatène les chaînes de caractères ch1 et ch2.
`part`(ch,v)	extraction des caractères de la chaîne de caractères ch dont le numéro est dans v
`string`(var)	conversion d'une expression numérique var en une chaîne de caractères
`convstr`(ch,opt)	pour passer la chaîne ch en minuscules ou majuscules suivant que opt='u' ou opt='l').
`strsub`	pour substituer une chaîne de caractères dans une autre.
`stripblanks`	pour supprimer les espaces dans l'expression.
`grep`(M,mot)	renvoie les indices des éléments de la matrice M contenant la chaîne mot.
`strindex`	pour trouver les positions d'une chaîne de caractères dans une autre.

E.2 L'évaluation d'une chaîne de caractères

`evstr` permet d'évaluer une expression écrite comme une chaîne de caractères.

Exemple 40.

```
--> ch = ['exp(a)', 'cos(b)'];
      a = 1; b = %pi;
--> evstr( ch )
```

```
ans  =
    2.7182818  - 1.
```

On peut l'utiliser en combinaison avec `feval` pour répéter les mêmes instructions sur plusieurs fonctions.

```
function y=f1(x)
    y=sin(x.^2);
endfunction

function y=f2(x)
    y=cos(x.^2);
endfunction
```

```
lfonct=['f1','f2'];
x=-3:0.1:3;
y=zeros(2,length(x));
for i=1:2
  y(i,:)=feval(x, evstr( lfonct(i) ));
end
```

Table des matières

1 Les caractéristiques principales **1**

2 L'accès **1**

3 L'environnement **1**
3.1 La fenêtre de commandes . . . 1
3.2 L'espace de travail . . . 3
3.3 L'éditeur de texte SciNotes . . . 5
3.4 L'aide en ligne . . . 5

4 Les types de données de base **5**
4.1 Les principaux types scalaires . . . 6
4.2 Le type matrice . . . 6
4.3 Création d'une matrice . . . 6
4.4 Extraction de valeurs . . . 7
4.5 Modification d'une matrice . . . 8

5 Les opérations sur les matrices **8**
5.1 Les opérations matricielles . . . 8
5.2 Les opérations sur chaque élément d'une matrice . . . 9
5.3 Quelques fonctions opérant sur une dimension de la matrice . . . 9

6 Opérateurs relationnels et logiques **10**
6.1 Représentations de points dans le plan . . . 11
6.2 La superposition de tracés . . . 12
6.3 Gestion de la fenêtre graphique et ajout de légendes . . . 12

7 Les fonctions et programmes **13**
7.1 Les fonctions . . . 13
7.2 Les programmes . . . 14

8 Les commandes structurées **16**
8.1 La boucle `for` . . . 16
8.2 L'instruction conditionnelle `if` . . . 17
8.3 La boucle conditionnelle `while` . . . 18
8.4 La programmation récursive . . . 19

9 Passer une fonction comme argument d'une autre fonction **19**

A Complément sur la manipulation des booléens **21**
A.1 Les fonctions `and` et `or` . . . 21
A.2 La fonction `find` . . . 21

B Conversion d'objets **21**

C L'analyse du fonctionnement d'une fonction **21**
C.1 Temps d'exécution d'un programme . . . 21
C.2 Le débogage d'une fonction . . . 22

D Complément sur la création de figure **22**
D.1 La superposition de tracés sur une même figure . . . 22
D.2 La sauvegarde d'une figure . . . 22
D.3 Les propriétés d'une figure . . . 22
D.4 Quelques fonctions pour la représentation graphique en 3D . . . 24

E Les chaînes de caractères **25**
E.1 La manipulation des chaînes de caractères . . . 25
E.2 L'évaluation d'une chaîne de caractères . . . 25