

COURS Scilab

Par Dimitri PIANETA

2016

Table des matières

A- INSTALLATION.....	3
B- La syntaxe	9
Chapitre 1 : la manipulation des matrices et vecteurs	10
Chapitre 2 : programmer Scilab.....	25
Chapitre 3 : graphes.....	31
Chapitre 4 : calcul numérique	36
Chapitre 5 : autres fonctions	38

A- INSTALLATION

I) Installation :

Scilab est un logiciel de calcul numérique que chacun peut télécharger gratuitement. Ce logiciel est recommandé au Lycée et à l'université pour remplacer le logiciel payant MATLAB.

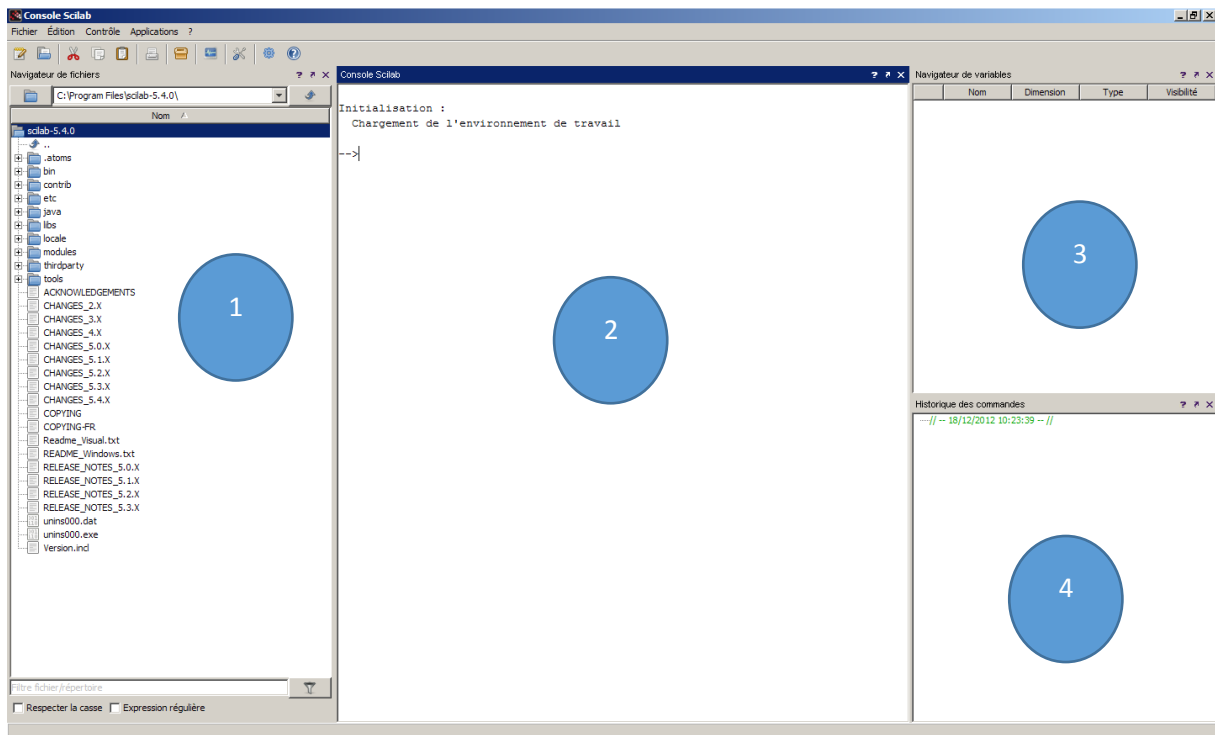
Disponible sous Windows, Linux et Max OS X.

On le télécharge à adresse suivante : <http://www.scilab.org/>

- Installer la version 5 car c'est la plus stable.
- Après installation supprimer la librairie java (jre qui est la JVM de java) si vous avez déjà installé une JRE par exemple jre8. Sinon rien faire.
- Lancer le programme puis interface de travail arrive.

II) Présentation IDE :

Scilab a créé un IDE de travail.



C'est la zone du chargement du répertoire de travail et des fichiers.



C'est la zone de commande (aussi appelé le terminal). Cette zone permet de faire des commandes appeler des fonctions, voir le résultat de calcul par exemple.



C'est la zone des variables, des tableaux.



C'est la zone de l'historique des commandes.

Comment effacer (2) et (4) ?

Le (2) efface soit en cliquant sur la zone (2) clique droite puis *effacer console*. Dans la console, on peut écrire `clc`.


Le (4) efface en cliquant sur la zone (2) clique droite puis *effacer historique*.

Pour interrompre un programme en cours d'exécution, on peut :

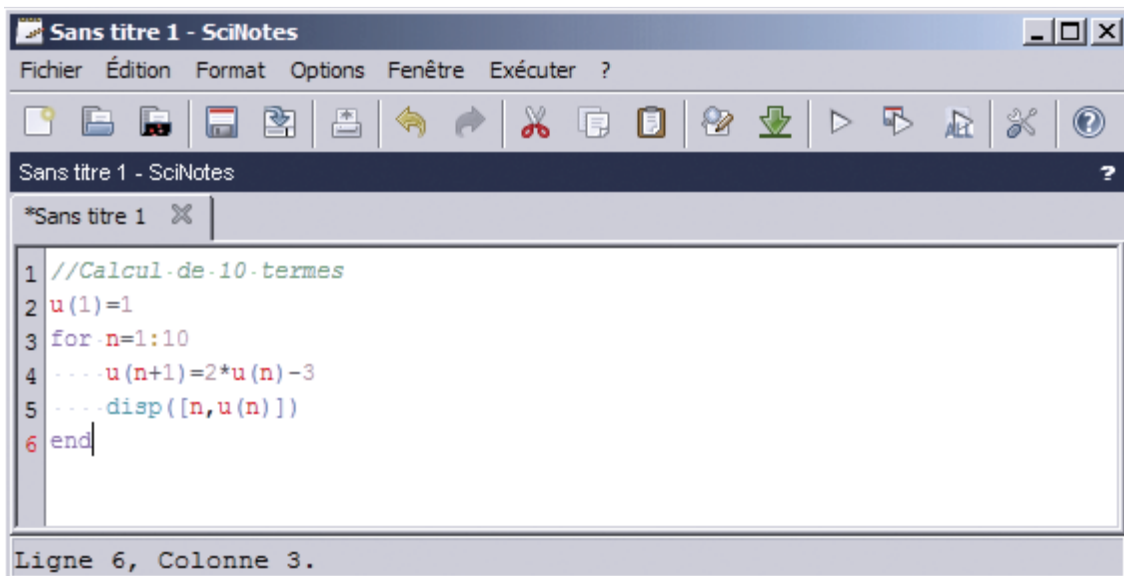
- Taper *pause* dans le programme ou cliquer sur *Contrôle>Interrompre* dans la barre de menus (ou CTRL +X), si le programme est déjà lancé. Dans tous les cas, l'invite de commande « --> » se transforme en « -1 », puis en « -2 », ... si l'opération est répétée.
- Pour revenir au moment de l'interruption du programme, taper *resume* dans la console ou cliquer sur *Contrôle>Reprendre*.
- Pour arrêter définitivement un calcul sans possibilité de retour, taper *abort* dans la console ou cliquer sur *Contrôle>Abandonner* dans la barre de menus.

III) Éditeur :

Taper directement dans la console a deux inconvénients : l'enregistrement n'est pas possible, et si plusieurs lignes d'instructions ont été tapées, les modifications ne sont pas aisées.

Pour ouvrir l'éditeur à partir de la console, cliquez sur la première icône  dans la barre d'outils ou sur **Applications>SciNotes** dans la barre de menus. (ou dans la console **edit**)

L'éditeur s'ouvre avec un fichier par défaut qui s'intitule « sans titre 1 ».



```
1 //Calcul de 10 termes
2 u(1)=1
3 for n=1:10
4 ... u(n+1)=2*u(n)-3
5 ... disp([n,u(n)])
6 end
```

POUR EXECUTER :

En cliquant sur **Exécuter** dans la barre de menus, trois options sont proposées :

- exécuter « **...fichier sans écho** » (Ctrl maj E sous Windows et Linux, Cmd maj E sous Mac OS X) : le fichier est exécuté sans que le programme ne s'écrive dans la console (en ayant enregistré le fichier au préalable).
- exécuter « **...fichier avec écho** » (Ctrl L sous Windows et Linux, Cmd L sous Mac OS X) : réécrit le fichier dans la console et l'exécute.
- exécuter « **...jusqu'au curseur, avec écho** » (Ctrl E sous Windows et linux, Cmd E sous Mac OS X) : réécrit la sélection choisie avec la souris dans la console et l'exécute

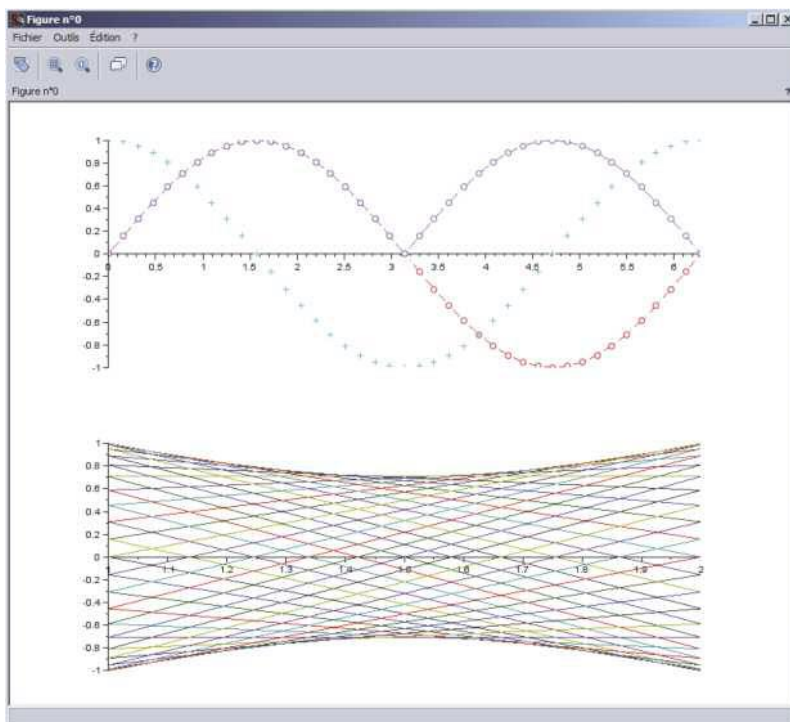
IV) La fenêtre Graphique :

Ouvrir une fenêtre graphique.

Une fenêtre graphique permet de tracer des courbes, des graphes....


On obtient un exemple de courbe en tapant dans la console :

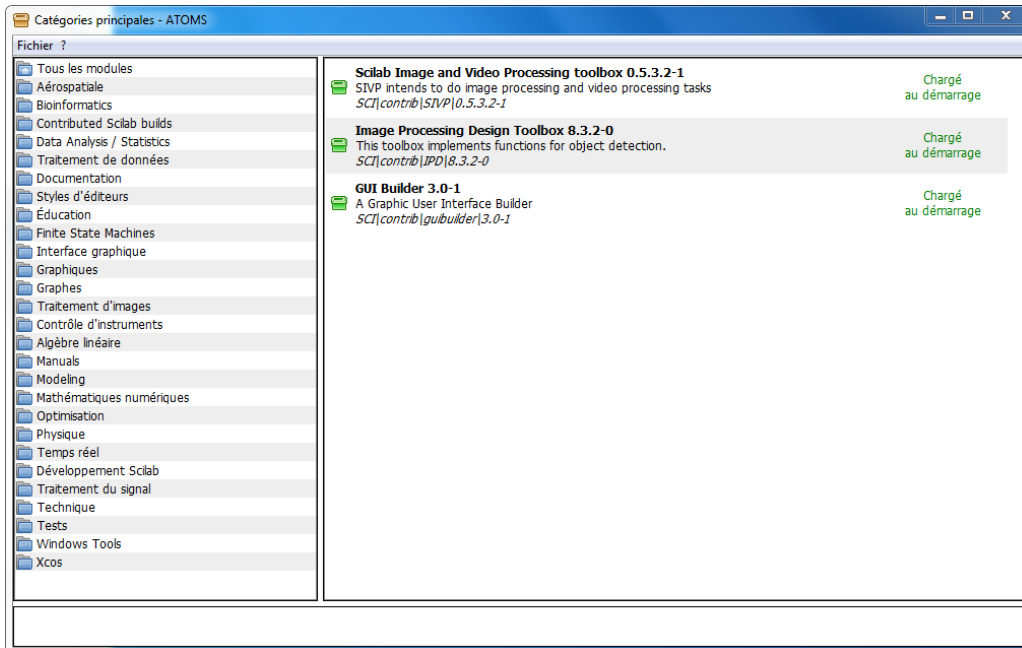
-->plot



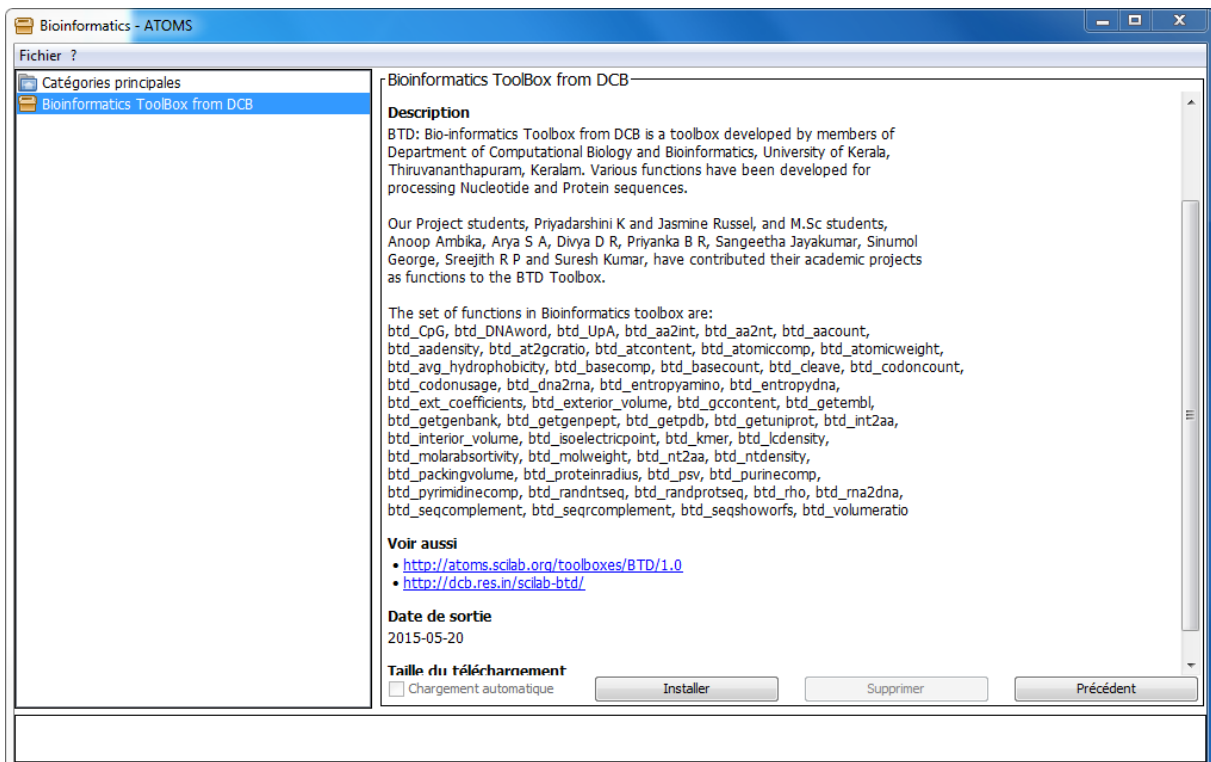
v) Installation toolbox :

Deux possibilités :

- Sur l'icône de la barre : 
- Soit dans le menu dans Application>Gestionnaire de module ATOMS

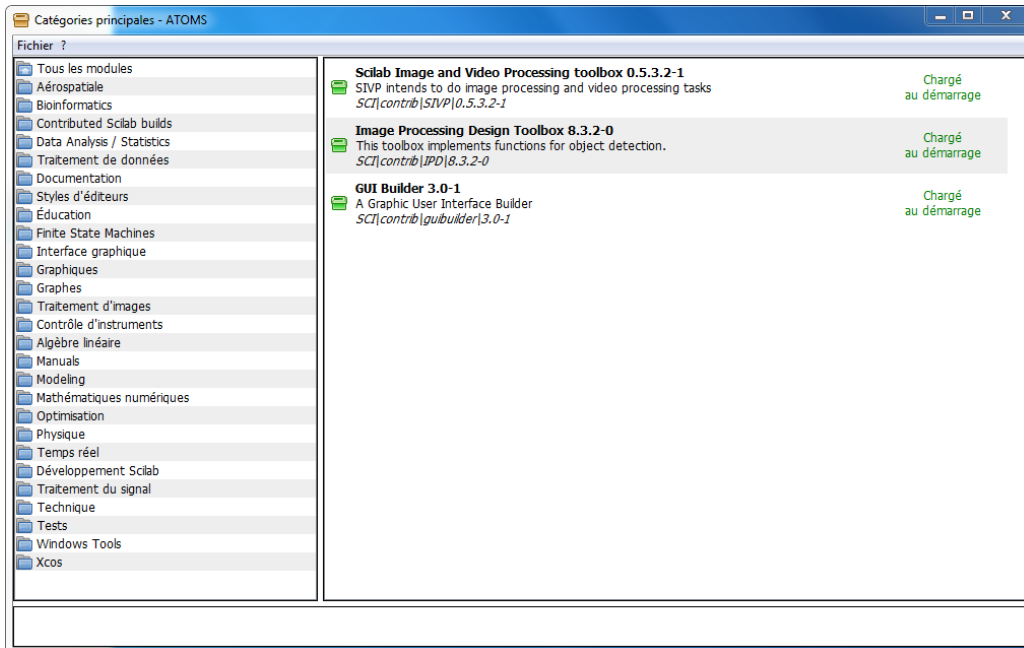


Choisir une rubrique que vous voulez puis par exemple :



Faites Installer ou Supprimer le toolbox installé

Puis sur cette figure, vous pouvez voir les modules installer



Puis redémarrer Scilab

B- La syntaxe

Chapitre 1 : la manipulation des matrices et vecteurs

a) Écrire une matrice :

```
A=[1 2 3; 4 5 6; 7 9 10]
```

```
--> A =
```

```
1. 2. 3.  
4. 5. 6.  
7. 9. 10.
```

b) Un vecteur :

```
b=[2 10 44 190];
```

```
-->b
```

```
b =
```

```
2. 10. 44. 190.
```

Attention : Une instruction très longue peut être écrite sur plusieurs lignes en écrivant trois points à la fin de chaque ligne à poursuivre :

```
-->T = [ 1 0 0 0 0 0 ;...  
--> 1 2 0 0 0 0 ;...  
--> 1 2 3 0 0 0 ;...  
--> 1 2 3 0 0 0 ;...  
--> 1 2 3 4 0 0 ;...  
--> 1 2 3 4 5 0 ;...  
--> 1 2 3 4 5 6 ]
```

c) Quelques matrices et vecteurs types

Matrice identité

Pour obtenir une matrice identité de dimension (4,4) :

```
-->I = eye(4,4)
```

```
I =
```

```
1. 0. 0. 0.  
0. 1. 0. 0.  
0. 0. 1. 0.  
0. 0. 0. 1.
```

Les arguments de la fonction `eye(n,m)` sont le nombre de lignes `n` et le nombre de colonnes `m` de la

matrice (Rmq : si $n < m$ (resp. $n > m$) on obtient la matrice de la surjection (resp. injection) canonique de \mathbb{K}^m vers \mathbb{K}^n .

Matrices diagonales, extraction de la diagonale

Pour obtenir une matrice diagonale, dont les éléments diagonaux sont formés à partir d'un vecteur :

```
-->B = diag(b)
```

```
B =
```

```
2. 0. 0. 0.
```

```
0. 10. 0. 0.
```

```
0. 0. 44. 0.
```

```
0. 0. 0. 190.
```

Remarque : cet exemple illustre le fait que Scilab distingue minuscule et majuscule, taper `b` pour vous rendre compte que ce vecteur existe toujours dans l'environnement.

Appliquée sur une matrice la fonction `diag` permet d'en extraire sa diagonale principale sous la forme d'un vecteur colonne :

```
-->b= diag(B)
```

```
b =
```

```
2.
```

```
10.
```

```
44.
```

```
190.
```

Syntaxe :

```
[y]=diag(vm, [k])
```

Avec

`vm` : vecteur ou matrice (stockage plein ou creux)

`k` : entier (valeur par défaut 0)

`y` : vecteur ou matrice

Matrices de zéros et de uns

Les fonctions `zeros` et `ones` permettent respectivement de créer des matrices nulles et des matrices « de 1 ». Comme pour la fonction `eye` leurs arguments sont le nombre de lignes puis de colonnes désirées.

Exemple :

```
-->C = ones(3,4)
```

C =

```
1. 1. 1. 1.
1. 1. 1. 1.
1. 1. 1. 1.
```

Mais on peut aussi utiliser comme argument le nom d'une matrice déjà définie dans l'environnement et tout se passe comme si l'on avait donné les deux dimensions de cette matrice :

```
-->O = zeros(C)
```

O =

```
0. 0. 0. 0.
0. 0. 0. 0.
0. 0. 0. 0.
```

Extractions des parties triangulaires supérieure et inférieure

Les fonctions `triu` et `tril` permettent elles d'extraire respectivement la partie triangulaire supérieure (u comme upper) et inférieure (l comme lower) d'une matrice, exemple :

```
-->U = triu(C)
```

U =

```
1. 1. 1. 1.
0. 1. 1. 1.
0. 0. 1. 1.
```

```
-->L = tril(C)
```

L =

```
1. 0. 0. 0.
1. 1. 0. 0.
1. 1. 1. 0.
```

Matrices de nombres aléatoires

La fonction rand (dont nous reparlerons) permet de créer des matrices remplies de nombres pseudo aléatoires (suivants une loi uniforme sur [0; 1[mais il est possible d'obtenir une loi normale et aussi de choisir le germe de la suite) :

```
-->M = rand(2,6)
M =
```

```
column 1 to 3
```

```
0.2113249  0.0002211  0.6653811
0.7560439  0.3303271  0.6283918
```

```
column 4 to 6
```

```
0.8497452  0.8782165  0.5608486
0.6857310  0.0683740  0.6623569
```

Autres syntaxes :

```
// Récupère un double aléatoire (sur la distribution courante)
r=rand()
```

```
// Récupère une matrice de doubles de taille 4-par-6 (sur la
distribution courante)
r=rand(4,6)
```

```
// Récupère une matrice de doubles de taille 4-par-6 de
distribution uniforme
r=rand(4,6,"uniform")
```

```
// Génère une matrice de doubles aléatoires normaux centrés
réduits de même taille que x
x=rand(4,4);
r=rand(x,"normal")
```

```
// Génère un tableau de taille 2-par-2-par-2 de doubles
aléatoires
r=rand(2,2,2)
```

Vecteurs à incrément constant entre 2 composantes

Pour rentrer un vecteur (ligne) x_a n composantes r_eguli_èrement r_eparties entre x1 et xn (c-a-d telles $x_{i+1} - x_i = \frac{x_n - x_1}{n-1}$, n « piquets » donc n-1 intervalles. . .), on utilise la fonction linspace :

```
--> x = linspace(0,1,11)
```

```
x =
```

```
column 1 to 7
```

```
0.  0.1  0.2  0.3  0.4  0.5  0.6
```

```
column 8 to 11
```

0.7 0.8 0.9 1.

Une instruction analogue permet, partant d'une valeur initiale pour la première composante, d'imposer « l'incrément » entre deux composantes, et de former ainsi les autres composantes du vecteur jusqu'à ne pas dépasser une certaine limite :

-->y = 0:0.3:1

y =

0. 0.3 0.6 0.9

La syntaxe est donc : y = valeur_initiale:incrément:limite_a_ne_pas_dépasser. Lorsque l'on travaille avec des entiers, il n'y a pas de problème (sauf entiers très grands...) à fixer la limite de sorte qu'elle corresponde à la dernière composante :

-->i = 0:2:12

i =

0. 2. 4. 6. 8. 10. 12.

Pour les réels (approché par des nombres flottants) c'est beaucoup moins évident du fait :

(i) que l'incrément peut ne pas « tomber juste » en binaire (par exemple $(0:2)_{10} = (0:00110011 : : :)_2$) et il y a donc un arrondi dans la représentation machine,

(ii) et des erreurs d'arrondi numérique qui s'accumulent au fur et à mesure du calcul des composantes.

Souvent l'incrément est égal à 1 et on peut alors l'omettre :

-->ind = 1:5

ind =

1. 2. 3. 4. 5.

d) Expressions possibles de faire :

-->A = {1 2 3; 4 5 6; 5 6 7}

A =

1. 2. 3.

4. 5. 6.

5. 6. 7.

Additions:

$$\rightarrow D = A + \text{ones}(A)$$

$$D =$$

$$2. \quad 3. \quad 4.$$

$$5. \quad 6. \quad 7.$$

$$6. \quad 7. \quad 8.$$

On peut qu'additionner des matrices carrées.

Multiplications :

$$\text{On pose } C = \{2 \ 1 \ 5 ; 6 \ 7 \ 8 ; 9 \ 10 \ 4\};$$

$$\rightarrow E = A * C$$

$$E =$$

$$41. \quad 45. \quad 33.$$

$$92. \quad 99. \quad 84.$$

$$109. \quad 117. \quad 101.$$

Transposée :

$$\rightarrow A_t = A'$$

$$A_t =$$

$$1. \quad 4. \quad 5.$$

$$2. \quad 5. \quad 6.$$

$$3. \quad 6. \quad 7.$$

Matrice de coefficient complexe :

$$\rightarrow A_c = A + \%i * \text{eye}(3,3)$$

$$A_c =$$

$$1. + i \quad 2. \quad 3.$$

$$4. \quad 5. + i \quad 6.$$

$$5. \quad 6. \quad 7. + i$$

Conjugué de matrice complexe :

$$\rightarrow A_{c_adj} = A_c'$$

$$A_{c_adj} =$$

- 1. - i 4. 5.
- 2. 5. - i 6.
- 3. 6. 7. - i

Un vecteur colonne :

-->x= linspace(0,1,5)'

x =

- 0.
- 0.25
- 0.5
- 0.75
- 1.

Un autre vecteur colonne :

->y = (1:5)'

y =

- 1.
- 2.
- 3.
- 4.
- 5.

Produit scalaire :

-->p = y'*x

p =

- 10.

e) Référencer, extraire, concaténer matrices et vecteurs :

Je rappelle A :

-->A

A =

- 1. 2. 3.
- 4. 5. 6.
- 5. 6. 7.

Il est possible de récupérer les valeurs dans la matrice.

On peut faire :

```
-->A33 = A(3,3)
```

A33 =

7.

Pour extraire par exemple la deuxième colonne, on écrit :

```
-->A(:,2)
```

ans =

2.

5.

6.

Pour extraire la troisième ligne :

```
-->A(3,:)
```

ans =

5. 6. 7.

Par exemple, on recherche la sous-matrice principale d'ordre 2

On effectue le calcul suivant :

```
-->A(1:2,1:2)
```

ans =

1. 2.

4. 5.

Passons maintenant à la syntaxe générale : si A est une matrice de taille $(n;m)$, et si $v1 = (i_1; i_2; \dots; i_p)$ et $v2 = (j_1; j_2; \dots; j_q)$ sont deux vecteurs (ligne ou colonne peut importe) d'indices dont les valeurs sont telles que $1 \leq i_k \leq n$ et $1 \leq j_k \leq m$ alors $A(v1,v2)$ est la matrice (de dimension $(p; q)$) formée par l'intersection des lignes $i_1; i_2; \dots; i_p$ et des colonnes $j_1; j_2; \dots; j_q$.

Exemples :

```
-->A([3,1],[2,1])
```

ans =

6. 5.

2. 1.

Dans la pratique on utilise généralement des extractions plus simples, comme celle d'un bloc contigu ou bien d'une (ou plusieurs) colonne(s) ou ligne(s). Dans ce cas, on utilise l'expression `i_debut:incr:i_fin` pour générer les vecteurs d'indices, ainsi que le caractère `:` pour désigner toute l'étendue dans la dimension adéquate. Ainsi pour obtenir la sous-matrice formée de la première et troisième ligne :

```
-->A(1:2:3,:)
```

ans =

```
1. 2. 3.
```

```
5. 6. 7.
```

Passons maintenant à la concaténation de matrices qui est l'opération permettant d'assembler (en les juxtaposant) plusieurs matrices, pour en obtenir une autre. Voici un exemple : on considère la matrice suivante, avec un découpage par blocs :

$$A = \left(\begin{array}{c|ccc} 1 & 2 & 3 & 4 \\ \hline 1 & 4 & 9 & 16 \\ 1 & 8 & 27 & 64 \\ 1 & 16 & 81 & 256 \end{array} \right) = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}.$$

```
-->A11 = 1
```

```
-->A12 = [2 3 4]
```

```
-->A21 = [1;1;1];
```

```
-->A22 = [4 9 16;8 27 64; 16 81 256];
```

```
-->A = [A11 A12 ; A21 A22]
```

A =

```
1. 2. 3. 4.
```

```
1. 4. 9. 16.
```

```
1. 8. 27. 64.
```

```
1. 16. 81. 256.
```

f) Quelques primitives matricielles supplémentaires :

- Somme, produit des coefficients d'une matrice, matrice vide

Pour faire la somme des coefficients d'une matrice, on utilise la méthode sum :

```
sum(1 :6)
```

```
ans = 21.
```

Cette fonction admet un argument supplémentaire pour effectuer la somme selon les lignes ou les colonnes :

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

```
B =
```

```
1. 2. 3.
```

```
4. 5. 6.
```

```
-->sum(B,"r") // on effectue la somme de chaque colonne -> on obtient une ligne
```

```
ans =
```

```
5. 7. 9.
```

```
-->sum(B,"c") // on effectue la somme de chaque ligne -> on obtient une colonne
```

```
ans =
```

```
6.
```

```
15.
```

De la même manière pour le produit des éléments d'une matrice, on dispose de la fonction prod :

```
-->prod(1:5)
```

```
ans =
```

```
120.
```

Rappelle B :

```
-->B
```

```
B =
```

```
1. 2. 3.
```

```
4. 5. 6.
```

```
-->prod(B,"r")
```

```
ans =
```

```
4. 10. 18.
```

```
-->prod(B,"c")
```

ans =

6.

120.

- somme et produit cumulés :

Les fonctions cumsum et cumprod calculent respectivement les sommes et produits cumulés d'un vecteur ou d'une matrice :

```
-->x = 1:6
```

x =

1. 2. 3. 4. 5. 6.

Le calcul de la somme cumulé :

```
-->cumsum(x)
```

ans =

1. 3. 6. 10. 15. 21.

Le calcul du produit cumulé :

```
-->cumprod(x)
```

ans =

1. 2. 6. 24. 120. 720.

Maintenant, je vais m'intéresser à calculer l'accumulation dans une matrice selon l'ordre colonne par colonne :

On pose x :

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

x =

1. 2. 3.

4. 5. 6.

```
-->cumsum(x)
```

ans =

1. 7. 15.

5. 12. 21.

On peut faire de même pour la fonction cumprod.

ET comme pour les fonctions sum et prod, on peut aussi faire les sommes et produits cumulés selon les lignes et les colonnes :

```
--> cumsum(x,"r")
```

```
ans =
```

```
1. 2. 3.
```

```
5. 7. 9.
```

```
--> cumsum(x,"c")
```

```
ans =
```

```
1. 3. 6.
```

```
4. 9. 15.
```

Donc $y = \text{cumsum}(x, "r")$ signifie la formule suivante :

$$y(i, :) = \sum_{k=1}^i x(k, :)$$

De même si $y = \text{cumsum}(x, "c")$ signifie la formule suivante :

$$y(:, j) = \sum_{k=1}^j x(:, k)$$

- Minimum et maximum d'un vecteur ou d'une matrice

Les fonctions min et max se chargent de ces opérations. Elles fonctionnent exactement comme sum et prod quand à l'argument supplémentaire pour calculer les minima ou maxima de chaque ligne ou colonne.

```
-->x = rand(1,5)
```

```
x =
```

```
column 1 to 3
```

```
0.2113249 0.7560439 0.0002211
```

```
column 4 to 5
```

```
0.3303271 0.6653811
```

```
-->x = min(x)
```

```
x =
```

```
0.0002211
```

-->[xmin, imin] = min(x)

imin =

1.

xmin =

0.0002211

Il est possible de rajouter un argument pour la fonction min soit « r » pour le calcul de chaque colonne et « c » pour le calcul de chaque ligne.

On fait deux mêmes que pour min pour la fonction max.

- **Moyenne et écart type :**

Les fonctions mean et st_deviation permettent de calculer la moyenne et l'écart type des composantes d'un vecteur ou d'une matrice. La formule utilisée pour écart type étant :

$$\sigma(x) = \left(\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2 \right)^{1/2}, \quad \bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

Syntaxe mean(x) et il est possible de calculer la moyenne des lignes ou des colonnes (« r » ou « c »)

Syntaxe st_deviation(x) et il est possible de calculer l'écart type des lignes ou des colonnes (« r » ou « c »)

- **Remodeler une matrice :**

La fonction matrix permet de remodeler une matrice en donnant de nouvelles dimensions.

-->B = [1 2 3; 4 5 6]

B =

1. 2. 3.

4. 5. 6.

-->B_new = matrix(B,3,2)

B_new =

1. 5.

4. 3.

2. 6.

- **Vecteurs avec espacement logarithmique :**

logspace(a,b,n) : permet d'obtenir un tel vecteur avec n composante, dont la première (a) et la dernière (b) sont respectivement 10^a et 10^b .

```
-->logspace(-2,5,8)
```

ans =

column 1 to 6

0.01 0.1 1. 10. 100. 1000.

column 7 to 8

10000. 100000.

- **Valeurs et vecteurs propres :**

La fonction spec permet de calculer les valeurs propres d'une matrice :

```
-->A= rand(5,5)
```

A =

column 1 to 3

0.2113249 0.6283918 0.5608486

0.7560439 0.8497452 0.6623569

0.0002211 0.6857310 0.7263507

0.3303271 0.8782165 0.1985144

0.6653811 0.0683740 0.5442573

column 4 to 5

0.2320748 0.3076091

0.2312237 0.9329616

0.2164633 0.2146008

0.8833888 0.312642

0.6525135 0.3616361

```
-->spec(A)
```

ans =

2.4777836

-0.0245759 + 0.5208514i

-0.0245759 - 0.5208514i

0.0696540

0.5341598

g) Les fonctions size et length :

La fonction size permet de récupérer les deux dimensions (nombre de lignes puis de colonnes) d'une matrice :

```
-->[nl,nc]=size(B) // B de l'exemple précédent
```

```
nc =
```

```
3.
```

```
nl =
```

```
2.
```

```
-->x=5:-1:1
```

```
x =
```

```
5. 4. 3. 2. 1.
```

```
-->size(x)
```

```
ans =
```

```
1. 5.
```

Alors length fournit le nombre d'éléments d'une matrice (réelle ou complexe). Ainsi pour un vecteur ligne ou colonne, on obtient directement son nombre de composantes :

```
-->length(x)
```

```
ans =
```

```
5.
```

Savoir le nombre de lignes d'une matrice : `size(A,'r')` ou `size(A,1)`

Savoir le nombre de colonnes d'une matrice : `size(A,'c')` ou `size(A,2)`

Chapitre 2 : programmer Scilab

I) Les boucles

I.1) La boucle for :

La boucle for itère sur les composantes d'un vecteur ligne :

```
-->v=[1 -1 1 -1]
```

```
-->y=0; for k=v, y=y+k, end
```

```
-->y=0; for i=1:4, y=y+v(i), end
```

I.2) La boucle while :

```
-->x=1 ; while x<14,x=2*x, end
```

Signalons que les opérations de comparaisons sont les suivants :

==	Égal à
<	Strictement plus petit que
>	Strictement plus grand que
<=	Plus petit ou égal
>=	Plus grand ou égal
~= ou <>	Différent de

II) Les instructions conditionnelles

II.1) La construction if then else

Voici un exemple :

```
->if x>0 then, y=-x,else,y=x,end
```

y =

- 16.

II.2) La construction du case

```
select num  
case 1 y = 'cas 1'  
case 2 y = 'cas 2'  
else y = 'autre cas'  
end
```

III) Les chaînes de caractères

On appelle chaîne de caractères, une suite de caractères.

Une chaîne de caractères en Scilab peut être écrite suivant avec les double guillemets ou simple guillemet.

Différentes méthodes peuvent être utilisées comme :

- **Pour obtenir les dimensions**

On pose le tableau suivant Ms= ["a" "bc" "def"]

On utilise la méthode suivante :

size(Ms) qui nous retourne un tableau de nombre de lignes 1 et nombre de colonnes 3.

- **Pour obtenir la longueur des mots dans un tableau par exemple**

On fait alors length(Ms) qui nous donne ans = 1. 2. 3. Qui signifie la longueur de gauche à droite des mots du tableau Ms.

- La concaténation d'une chaîne de caractères se fait simplement avec opérateur +.

Ex : s1 = 'abc'; s2 = 'def'; s = s1 + s2

s =
abcdef

Et l'extraction se fait via la fonction part :

-->part(s,3)

ans =

c

-->part(s,3:4)

ans =

cd

IV) Les listes

Une liste est une collection d'objets Scilab numérotés. Il y a deux sortes de listes, les ordinaires et les typés.

Liste ordinaire :

--> L=list(rand(2,2),["Vivement que je finisse" " cette doc..."],[%t ; %f])

L =

L(1)

0.2113249 0.0002211

0.7560439 0.3303271

L(2)

!Vivement que je finisse cette doc... !

L(3)

T

F

Extraction de la première entrée

```
-->M = L(1)
```

M =

```
0.2113249 0.0002211
```

```
0.7560439 0.3303271
```

Modification de la première entrée

```
-->L(1)(2,2) = 100;
```

```
-->L(1)
```

```
0.2113249 0.0002211
```

```
0.7560439 100.
```

On peut de même utilisés les méthodes size et length.

Liste typés :

On pose :

```
->P=[ 0 0 1 1 0 0 1 1;... // les coordonnees des sommets
```

```
-->0 1 1 0 0 1 1 0;...
```

```
-->0 0 0 0 1 1 1 1];
```

Et

```
-->connect = [1 5 3 2 1 1;... // la connectivite des faces
```

```
-->2 8 7 6 5 4;...
```

```
-->3 7 8 7 6 8;...
```

```
-->4 6 4 3 2 5];
```

```
-->Cube = tlist(["polyedre","coord","face"],P,connect)
```

Ce qui nous donne :

Cube =

Cube(1)

!polyedre coord face !

Cube(2)

```
0. 0. 1. 1. 0. 0. 1. 1.
0. 1. 1. 0. 0. 1. 1. 0.
0. 0. 0. 0. 1. 1. 1. 1.
```

Cube(3)

```
1. 5. 3. 2. 1. 1.
2. 8. 7. 6. 5. 4.
3. 7. 8. 7. 6. 8.
4. 6. 4. 3. 2. 5.
```

Au lieu de designer les éléments constitutifs par leur numéro, on peut utiliser la chaîne de caractères correspondante, exemple :

```
-->Cube.coord(:,2)
```

ans =

```
0.
1.
0.
```

Ou

```
-->Cube("coord")( :,2)
```

ans =

0.

1.

0.

-->Cube.face(:,1)

ans =

1.

2.

3.

4.

V) Les fonctions

Pour définir une fonction en Scilab, la méthode la plus courante est de l'écrire dans un fichier, dans lequel on pourra d'ailleurs mettre plusieurs fonctions. Chaque fonction doit commencer par l'instruction :

```
function [y1, y2, y3, ..., yn] = nomfonction(x1, ..., xm)
```

Premier exemple de fonction la factorielle :

```
function [y] = fact1(n)
    //la factorielle : il faut ici n soit en entier natrueel
    y = prod(1:n)
endfunction
```

Supposons que l'on ait écrit cette fonction facts.sci. Pour que Scilab puisse la connaître, il faut charger le fichier par l'instruction :

```
exec('C:\Users\Dimitri\Desktop\essai scilab\facts.sci', -1)
```

On fait ensuite :

```
-->m = fact1(5)
```

m =

120.

Second exemple : l'équation du second degré

```
function [x1, x2]=resoud_equ_2d(a, b, c)
    // calcul des racines de  $a x^2 + bx + c = 0$ 
    // a, b et c peuvent être des reels ou des complexes et a doit être non nul
    delta = b^2-4*a*c
    x1 = (-b-sqrt(delta))/(2*a)
    x2 = (-b+sqrt(delta))/(2*a)
endfunction
```

Voici trois essais :

```
-->[r1, r2] = resoud_equ_2d(1,2,1)
```

```
r2 =
```

```
- 1.
```

```
r1 =
```

```
- 1.
```

```
-->[r1, r2] = resoud_equ_2d(1,0,1)
```

```
r2 =
```

```
i
```

```
r1 =
```

```
- i
```

```
-->resoud_equ_2d(1,0,1)
```

```
ans =
```

```
- i
```

VI) L'instruction break

Pour terminer une condition

VII) Types de données

Constantes prédéfinies :

%pi 3.1415927

%e 2.7182818

%i $\sqrt{-1}$

%eps précision machine

%inf infini

%t vrai

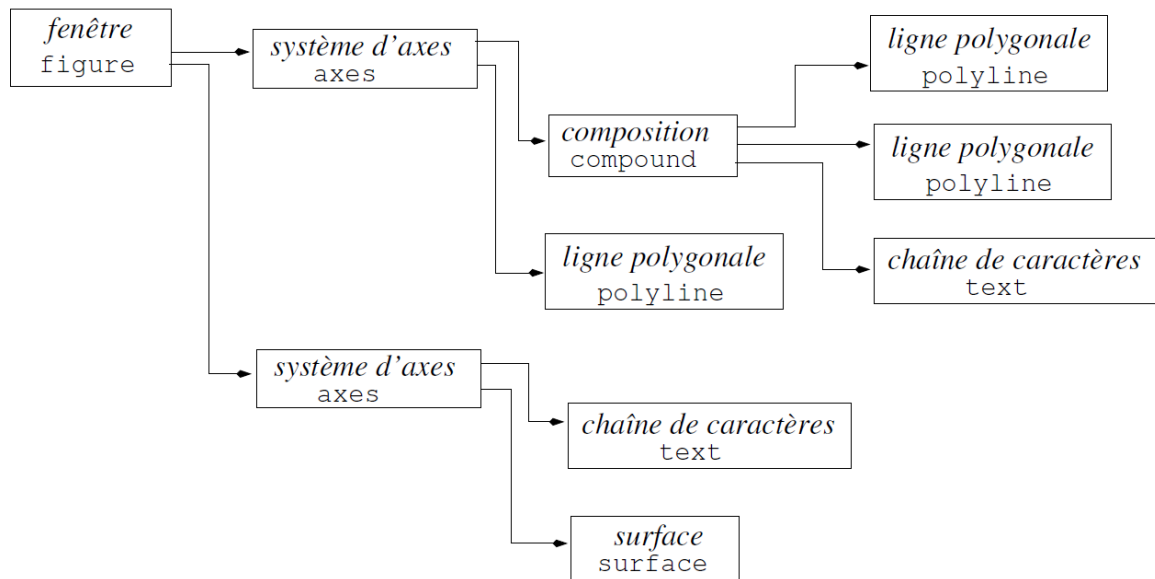
%f faux

%s variable de polynôme

Chapitre 3 : graphes

I) Schéma de principe des graphes

Pour faire un graphe, il faut construire une fenêtre qui prendra différents composants comme le graphe ci-dessous.



II) Les mots clés :

scf(num)	La fenêtre courante devient la fenêtre de numéro num ; si cette fenêtre n'existait pas, elle créée par Scilab.
xselect()	Met en « avant » la fenêtre courante ; Si aucune fenêtre graphique n'existe, Scilab en crée une.
clf(num)	Efface la fenêtre graphique numéro num ; Si num est omis, Scilab efface la fenêtre courante.
xdel([num])	Détruit la fenêtre graphique numéro num ; Si num est omis, Scilab détruit la fenêtre courante.

a) L'instruction plot

Par exemple cette suite instruction :

```
x = linspace(-1,1,61) ;
```

```
y = x.^2
```

```
plot(x,y)
```

Il est possible d'ajouter un **titre** avec `xtitle("Courbes...")` et une légende : `legend("y=x^2")`.

Les propriétés pour changer la couleur du trait, la forme.

Les couleurs

k	noir	c	Cyan
b	Bleu	m	Magenta
r	Rouge	y	Jaune
g	Vert	w	Blanc

Les types de traits

-	Trait plein
--	tirets
:	pointillés
-.	Tiret-point, etc

Les symboles

+	+	^	△	s	□
x	×	v	▽	.	.
o	○	>	▷	*	*
d	◇	<	◁	pentagram	★

Exemple de syntaxe :

```
plot(x,y,"b--",x,ybis,"ro",x,yter,"gx")
```

Modification le placement des axes

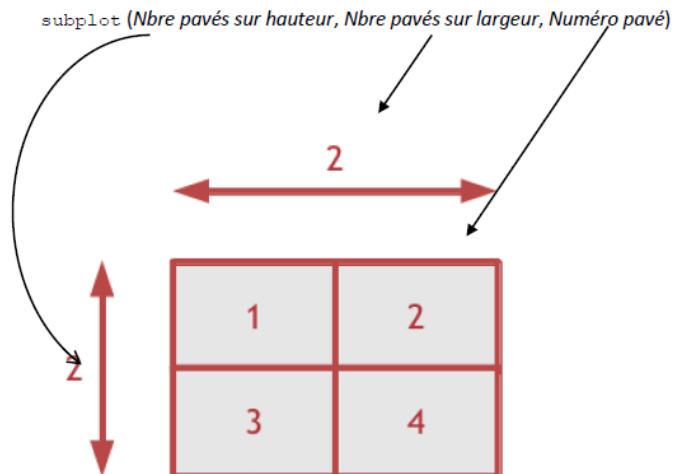
Cela s'obtient en jouant avec les propriétés x location et y location du système d'axes contenant le graphique.

x_location	placement obtenu	y_location	placement obtenu
"bottom"	en bas	"left"	à gauche bas
"top"	en haut	"right"	à droite
"middle"	en $y = 0$ si possible	"middle"	en $x = 0$ si possible

Voici un exemple :

```
x = linspace(-14,14,300);
y = sinc(x);
clf()
plot(x, y, "b");
a = gca();
a.x_location = "middle"; a.y_location = "middle";
a.title.text = "La fonction sinus cardinal";
```


b) L'instruction subplot



Exemple :

```
x = 0: 2*pi/100 : 2*pi;
subplot(221)
plot(x,sin(x))
subplot(222)
plot(x,cos(x),x,sin(x),'-.')
subplot(223)
plot(cos(x),sin(x))
subplot(224)
plot(sin(2*x),sin(3*x))
```

Modifier les propriétés d'une courbe : pour cela il faut récupérer le handle de la courbe qui vous intéresse ce qui n'est pas forcément évident. Une instruction comme `plot` « enrobe » ses courbes dans un objet `composition` dans l'ordre inverse de celui que l'on a donnée au départ. Les courbes sont donc filles de cette composition. D'autre part, à l'instar de la fenêtre courante et du système d'axes courant, il existe aussi la notion d'entité courante et justement, après un `plot`, l'entité courante est la composition de nos courbes. Ces quelques explications devraient vous aider à comprendre les manipulations suivantes :

```
x = linspace(0,15,200)';
y = besselj(1:4,x);
clf()
drawlater()
plot(x,y(:,1),"b", x,y(:,2),"r", x,y(:,3),"g", x,y(:,4),"c")
e = gce(); // l'identificateur de la composition des 4 courbes
// (le handle du compound des 4 courbes)
e.children(1).thickness = 3; // on change l'épaisseur de la
// courbe en cyan (la 4 eme)
e.children(2).foreground = 19; // on change la couleur de la
// courbe initialement verte (la 3 eme)
e.children(3).line_style = 2; // on change le style de la
// courbe en rouge (la 2 eme)
legend("J"+string(1:4));
```

```
xtitle("Quelques fonctions de Bessel")
xgrid(12)
drawnow()
```

c) L'instruction plot2d

plot2d(abcisses,ordonnees,style,cadre,bornes,graduation)

- **Abscisses, ordonnees** : ce sont nécessairement des matrices de mêmes dimensions. Si ce sont des vecteurs, ils peuvent être ligne ou colonne. Si plusieurs courbes doivent être tracées, elles doivent correspondre à autant de colonnes. À chaque courbe correspond une couleur de la palette (il y en a 32).
- **Style** : c'est un vecteur ligne dont la dimension est le nombre de courbes à tracer (nombre de colonnes des matrices abcisses et ordonnées). Les coordonnées sont positives ou négatives. Si le style est nul, les points sont affichés comme des pixels noirs. Si le style est négatif, des marques de formes particulières sont affichées.
- **Cadre** : ce paramètre est une chaîne de caractères formée de trois chiffres, dont :
 - o Le premier code la présence de légendes (0 ou 1) ;
 - o Le deuxième code le calcul des échelles en abscisse et ordonnée ;
 - o Le troisième code le tracé des axes ou du cadre.

Par défaut le cadre vaut « 021 » ce qui signifie (pas de légendes, échelles de représentation calculées automatiquement, axes tracés). Si l'on superpose deux graphiques avec cette option par défaut, les échelles ne seront pas les mêmes. La solution consiste à tracer tous les graphiques à partir du second sur une même fenêtre avec l'option « 000 » (pas de légende, utiliser les échelles précédentes, ne pas retracer les axes).

- **Légendes** : c'est une chaîne de caractères contenant les différentes légendes, séparées par @.

Ex : legendes = " x sin(x)@2 sin(x)@3"

- **Bornes** : c'est le rectangle de représentation, décrit par les deux coordonnées du coin inférieur gauche, suivies des deux coordonnées du coin inférieur droit : [xmin,ymin,xmax,ymax].
- **Graduations** : ce vecteur de quatre entiers permet de préciser la fréquence des graduations et sous-graduations en abscisse et ordonnée.

On a alors différents plot :

```
plotframe  rectangle de représentation
plot      points joints par des segments
plot2d    plusieurs courbes avec styles différents
plot2d1   idem, avec plus d'options
plot2d2   représentation en escalier
plot2d3   barres verticales
plot2d4   flèches
fplot2d   représenter des fonctions
```

d) Graphiques composites:

`xbasf();`

`xset("font",2,4);`

...

Ajouts sur graphique

xarc arc d'ellipse

xfarc arc d'ellipse plein

xarrows flèches

xnumb nombres

xpoly polygone

xfpoly polygone plein

xrpoly polygone régulier

xrect rectangle

xfrect rectangle plein

xstring chaîne de caractères (à partir d'un point)

xstringb chaîne de caractères (dans un rectangle)

xtitle titre du graphique et des axes

Représentations planes particulières

histplot histogramme

champ champ de vecteurs

fchamp idem, définition par une fonction

grayplot surface par rectangles de couleurs

fgrayplot idem, définition par une fonction

contour2d courbes de niveaux projetées

fcontour2d idem, définition par une fonction

e) Dimension 3

param3d courbes paramétriques

param3d1 plusieurs courbes ou points

plot3d surface en dimension 3

fplot3d idem, définition par une fonction

plot3d1 surface par niveaux de couleur

fplot3d1 idem, définition par une fonction

eval3dp surface paramétrée

hist3d histogramme

Chapitre 4 : calcul numérique

Algèbre linéaire :

A' transposée de A
rank rang
inv inverse
expm exponentielle matricielle
det déterminant
trace trace
poly(A,"x") polynôme caractéristique de A
spec valeurs propres de A
bdiag diagonalisation
svd décomposition en valeurs singulières
A\b solution de $A*x=b$
b/A solution de $x*A=b$
linsolve(A,b) solution de $A*x=-b$

Intégration :

Les fonctions `integrate`, `intg`, `int2d`, `int3d`, `intc` et `intl` prennent en entrée une fonction externe, ou définie par une chaîne de caractères. Les fonctions `integ`, `inttrap` et `intsplin` prennent en entrée des vecteurs d'abscisses et ordonnées.

integrate fonction définie par un chaîne de caractères
intg fonction externe
integ vecteurs abscisses et d'ordonnées
inttrap méthode des trapèzes
intsplin approximation par splines
int2d fonction de deux variables
int3d fonction de trois variables
intc fonction complexe le long d'un segment
intl fonction complexe le long d'un arc de cercle

Transformées :

dft transformée de Fourier discrète
fft transformée de Fourier rapide
convol produit de convolution
flt transformée de Legendre rapide
dmt transformée de Mellin discrète
cwt transformée en ondelette continue

Résolutions :

fsolve systèmes d'équations
roots racines d'un polynôme

factors facteurs irréductibles réels d'un polynôme
linsolve systèmes linéaires

Optimisation :

optim optimisation
limpro programme linéaire
quapro programmation quadratique

Polynômes :

poly(v, « x ») polynôme dont les racines sont les éléments de v
plot(v, »x », « c ») polynôme dont les coefficients sont les éléments de v
inv_coeff(v) idem
coeff(P) coefficients du polynôme P
roots(P) racines du polynôme de P
factors facteurs irréductibles réels d'un polynôme

Chaînes de caractères :

evstr évaluer une expression
deff définir une fonction
execstr exécuter une instruction
length longueur
part extraire
+ concaténer
string transformer en chaîne

Chapitre 5 : autres fonctions

Aide

help() démarre le système d'aide

help (*str_cmd*) recherche l'aide sur la fonction *cmd*

--> help('inv')

apropos(*str_kwd*) recherche dans l'aide le mot-clé *kwd* et ordonne par pertinence les résultats

--> apropos('inv')

Environnement

who liste les variables connues dans l'environnement.

whos liste de manière détaillée les variables connues dans l'environnement.

clear(*str_var1, str_var2, ..., str_varn*) supprime les variables

var1, var2, ..., varn de l'environnement.

--> clear('M','n','i')

clear() supprime toutes les variables.

load(*str_fichier*) charge les variables sauvegardées dans *fichier*.

save(*str_fichier*) sauvegarde l'ensemble des variables dans *fichier*.

c_id = **diary**(*str_fichier*) ouvre le journal *id* dans le fichier et sauvegarde les commandes entrées dans la console.

diary(*c_id, 'close'*) sauve le journal *id*.

--> id = diary('TP1.txt')

id =

1.

--> // des commandes

--> diary(id, 'close')

ls() liste des fichiers du répertoire de travail.

pwd() affiche le répertoire courant.

cd(*str_rep*) modifie le répertoire courant (-> *rep*).

exec (*str_script*) exécute le fichier de commandes *script*.

--> exec('TP1.sce')

scinotes() lance l'éditeur de texte intégré de Scilab

Transformations

flipdim(*v, 1*) inverse l'ordre des éléments du vecteur *v*.

flipdim(*M, c_dim*) retourne la matrice *M* selon la dimension *dim*.

matrix(*v, c_lig, c_col*) retourne une matrice de taille *lig* x *col* à partir des valeurs de *v*.

matrix(*M, c_lig, c_col*) ou **matrix**(*M, [c_lig, c_col]*) retourne une matrice *lig* x *col* à partir des valeurs de *M*.

--> v=1:6; M = matrix(v,2,3)

ans =

1. 3 5.

2. 4. 6.

--> matrix(M,3,2)

ans =

1. 4.

2. 5.

3. 6.

repmat(*M,c_lig, c_col*) ou **repmat**(*M,[c_lig, c_col]*)

retourne une matrice (*lig * n*)x(*col*m*) par recopie de la matrice *M* de taille *nx m*.

-->M = [1 2 ; 3 4]

M =

1. 2.

3. 4.

-->repmat(M,2,3)

ans =

1. 2. 1. 2. 1. 2.

3. 4. 3. 4. 3. 4.

1 2. 1. 2. 1. 2.

3. 4. 3. 4. 3. 4.

Autres fonctions utiles

find(*v*) retourne les indices de valeurs de *v* différentes de 0.

find(*M*) retourne les indices linéaires de valeurs de *M* différentes de 0.

[*v_l,v_j*] = **find**(*M*) retourne les indices ligne *l* et colonne *J* des valeurs de *M* différentes de 0.

-->v = [2, 0, 1, 0, -1, 3]

v =

2. 0. 1. 0. - 1. 3

-->find(v)

ans =

1. 3. 5. 6.

-->find(matrix(v,2,3))

ans =

1. 3. 5. 6.

-->[I,J] = find(matrix(v,2,3))

J =

1. 2. 3. 3.

I =

1. 1. 1. 2.

find(*v_b*), **find**(*M_b*), [*v_l,v_j*] = **find**(*M_b*) idem pour les vecteurs et matrices booléennes avec les valeurs %t.

and(*v_b*) retourne %t ssi tous les éléments de *b* sont %t (quantificateur universel &).

and(*v_b*) retourne %t ssi tous les éléments de *b* sont différents de 0.

or(*v_b*) retourne %t ssi au moins un élément de *b* est %t (quantificateur existentiel &).

or(*v_b*) retourne %t ssi au moins un élément de *b* est différent de 0.

-->v = [2, 0, 1, 0, -1, 3]

v =

2. 0. 1. 0. - 1. 3.

-->v>0

ans =

T F T F F T

-->[and(v>0) or(v<0)]

ans =

F T

Programmation

Fonctions et paramètres

```
function [var_o1,...,var_om]=nom_fonct(var_i1,...,var_in)
cmd_1
```

...

```
cmd_n ;
```

```
endfunction
```

où i_1, \dots, i_n et o_1, \dots, o_m désignent respectivement les paramètres d'entrée et les sorties de la fonction *nom_fonc*. *Convention* : le code de la fonction *nom_fonc* se trouve dans le fichier *nom_fonc.sci*.

Extrait du fichier "mafonction.sci"

```
function [x2]=mafonction(x)
```

```
x2 = x**2;
```

```
endfunction
```

Puis dans la console

```
-->mafonction(3)
```

```
ans =
```

```
9.
```

deff(*str_specif*, *str_code*) permet de définir une fonction (courte) en ligne sans passer par un fichier .sci. La chaîne de caractères *str_specif* contient la spécification de la fonction, la chaîne de caractères *str_code* en contient le code.

```
-->deff('[x2]=mafonction(x)', 'x2 = x**2');
```

```
-->mafonction(3)
```

```
ans =
```

```
9.
```

deff(*str_specif*, *str_code*, 'p') rajoute l'option de profilage à la fonction définie en ligne.

Structures de contrôle

Branchement conditionnel **if**

```
if cond then
```

```
cmd_1 ; ... ; cmd_n ;
```

```
    elseif cond_i then
```

```
cmd_i1 ; ... ; cmd_in ;
```

```
    else
```

```
cmd_e1 ; ... ; cmd_en ;
```

```
end
```

Répétitive **for**

```
for var=v do
```

```
cmd_1 ; ... ; cmd_n ;
```

```
end
```

```
for var=M do
```

```
cmd_1 ; ... ; cmd_n ;
```

```
end
```

Répétitive **while**

```
while cond do
```

```
cmd_1 ; ... ; cmd_n ;
```



```

    end
v = 1;
    if v==0,
        for i=1:3,
            disp(i);
        end
    else
        i=3;
        while i<=3,
            disp(i);
            i = i+1;
        end,
    end
end

```

Autres structures de données

struct(*str_chp1, var1, ..., str_chpn, varn*) retourne un enregistrement composé d'un ensemble de couples champ / valeur.

```

-->M = rand(26,2);M = M.' * M;
-->S = struct('Mat',M,'Det',det(M))
S =
Mat: [2x2 constant]
Det: 21.51
-->S.Mat
ans =
7.54 7.885
7.885 11.1
-->S.det= 6;

```

list(*var1, var2, ..., varn*) est une liste composée des éléments passé en paramètres.

```

-->L = list(1:5,rand(2,2),[%f %f %t])
L =
L(1)
1. 2. 3. 4. 5.
L(2)
0.560 0.728
0.125 0.268
L(3)
F F T
-->for l=L,
-->disp(sum(l))
-->end
15.
1.68
1.
-->L(4) = 'chaine';

```

Flot d'exécution, Débogage

break interrompt la répétitive courante.

`[var_o1,...,var_om] = return(var1,...,varm)` provoque la sortie inconditionnelle de la fonction en cours d'interprétation.

pause interrompt l'exécution en l'attente de l'appui sur le clavier.

xpause(*c_tps*) interrompt l'exécution pendant *tps* millisecondes.

setbpt(*str_func*), **setbpt**(*str_func,c_lig*) ajoute un point d'arrêt à l'entrée de la fonction *nunc* ou à un certain numéro *lig* de ligne.

delbpt(), **delbpt**(*str_func*), **setbpt**(*str_func,c_lig*) supprime toute ou partie des points d'arrêt existants.

dispbpt() liste les points d'arrêt existants.

abort interrompt l'interprétation courante.

quit provoque la sortie de Scilab.

tic() démarre le chronomètre.

toc() retourne le temps écoulé depuis le dernier **tic**.

showprofile(*var_fun*) décore un code des informations de profilage après exécution de la fonction *fun*.

plotprofile(*var_fun*) affiche un graphique de profilage après exécution de la fonction *fun*.

```
deff('benchchol(n)', ['for i=1:n'  
' M = rand(i,i)'  
' chol(M.'*M)'  
'end'], 'p')  
-->benchchol(500);  
-->showprofile(benchchol)  
function []=fun(n) |1 |0 |0|  
for i = 1:n, |500|0 |0|  
M = rand(i, i) |500|0.96|4|  
chol((M.') * M)|500|9.24|6|  
end, |1 |0 |0|  
endfunction |1 |0 |0|
```

7

Opérations mathématiques :

abs	Valeur absolue ou module
exp	Exponentielle
log	Logarithme népérien
log10	Logarithme base de 10
os	Cosinus (argument en radian)
sin	Sinus (argument en radian)
sinc	Sin(x)/x
tan	Tangente (argument en radian)
cotg	Cotangente (argument en radian)
acos	Arcos (arc cosinus)
asin	arcsin
atan	arctan
cosh	Ch (cosinus hyperbolique)
sinh	Sh
tanh	Th
acosh	Argch (argument hyperbolique cosinus)
asinh	Argsh
atanh	Argth
sqrt	Racine carrée
floor	Partie entière $E(x) = ([x]) = n \Leftrightarrow n \leq x < n+1$
ceil	Partie entière supérieure $[x] = n \Leftrightarrow n-1 < x \leq n$
int	Partie entière anglaise $\text{int}(x) = [x]$ si $x > 0$ et $[x]$ sinon
erf	Fonction erreur $\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$
erfc	Fonction erreur complémentaire $\text{erfc}(x) = 1 - \text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_x^{+\infty} e^{-t^2} dt$
gamma	$\Gamma(x) = \int_0^{+\infty} t^{x-1} e^{-t} dt$
lngamma	$\ln(\Gamma(x))$
dlgamma	$\frac{d}{dx} \ln(\Gamma(x))$

Information sur l'espace de travail :

→ who