PETIT MEMENTO SCILAB

FRANÇOIS DUCROT

1. manipulation de vecteurs et matrices

1.1. **Création de matrices.** D'abord quelques briques élémentaires utiles pour construire des choses plus compliquées.

`1:4.5`	nombres compris entre 1 et 4.5 par incrément de 1
`1:1.5:3`	nombres compris entre 1 et 3 par incrément de 1,5
`linspace(a,b,n)`	vecteur constitué de n nombres régulièrement espacés entre a et b
`[]`	matrice vide
`[1,4,3]`	vecteur ligne $(1, 4, 3)$
`[1,2;3,4]`	matrice $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$
`ones(3,4)`	matrice de taille 3×4 remplie de 1
`zeros(3,4)`	matrice de taille 3×4 remplie de 0
`eye(5,4)`	matrice de taille 5×4 avec des 1 sur la diagonale et des 0 ailleurs
`diag(x)`	matrice carrée diagonale de diagonale le vecteur x
`diag(x,3)`	matrice carrée de i-ème diagonale diagonale le vecteur x
`toeplitz(x)`	matrice de toeplitz bâtie sur le vecteur x
`rand(m,n)`	matrice aléatoire (loin uniforme sur $[0, 1]$) de taille $m \times n$

1.2. **Opérations sur les matrices.** On peut effectuer des opérations sur les matrices :

`A'`	transposée conjuguée de A (transposée dans le cas réel)
`A+B`	somme de deux matrices de mêmes tailles
`A*B`	produit de deux matrices de tailles compatibles
`5*A`	5 fois la matrice A
`A^n`	puissance n-ième d'une matrice carrée A
`A.*B`	produit coefficient par coefficient de deux matrices de mêmes tailles
`A.^n`	matrice constituée des puissances n-ièmes des coefficients de A

1.3. **Concaténation de tableaux.** Un exemple suffira :
`A=(1;2;3) , B=eye(3,3) , C=ones(1,4), X=[A,B;C]`
donne le résultat

$$A = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \ , \ B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \ , \ C = (1, 1, 1, 1) \ , \ X = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 2 & 0 & 1 & 0 \\ 3 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Date: Septembre 2006.

1.4. **Extraction d'une partie d'un tableau.** On rentre d'abord une matrice
`M=[1,2,3;4,5,6;7,8,9]`
dont extrait ensuite différentes parties :

`M(2,3)`	élément $M_{2,3}$
`M(:,1)`	première colonne
`M(1,:)`	première ligne
`M([2,3],:)`	matrice constituée des 2ième et 3ième lignes
`M([1,3],[1,2])`	matrice $\begin{pmatrix} 1 & 2 \\ 7 & 8 \end{pmatrix}$

Et en combinant extraction et affectations :

`M(3,3)=0`	l'élément $M_{3,3}$ de M devient 0, le reste est inchangé
`M(1,:)=2*M(1,:)`	la première ligne est multipliée par 2
`M(:,[1,2])=M(:,[2,1])`	les colonnes 1 et 2 sont échangées
`M($,:)=[]`	remplace la dernière ligne par une ligne vide. La taille de M devient 2×3

2. Quelques fonctions numériques ou vectorielles

Fonctions trigonométriques : `sin`, `cos`, `tan`
Fonction trigonométriques inverses : `asin`, `acos`, `atan`
Exponentielle, logarithme néperien ou décimal : `exp`, `log`, `log10`
Racine carrée : `sqrt`
valeur absolue, partie entière, signe : `abs`, `floor`, `sign`vspace5mm
Somme des coefficients d'un vecteur ou d'une matrice : `sum`
Produit des coefficients d'un vecteur ou d'une matrice : `prod`
Réordonner les coefficients d'un vecteura par ordre décroissant : `sort(a)`. La commande `gsort(a)` fait la même chose, mais offre des arguments supplémentaires.
Norme euclidienne d'un vecteur : `norm`
Déterminant d'une matrice : `det`
Solution du système matriciel $Ax = b$: `A\b`

3. Fonctions à valeurs logiques

Scilab connait deux constantes booléennes `%t` et `%f` (vrai et faux). On dispose alors des fonctions boolénennes usuelles `%==`, `%<`, `%>`, `%>=`, `%<=`, qu'on peut aussi relier par des connecteurs logiques `%~` (non), `%|` (ou), `%&`. On peut appliquer certaines de ces fonctions à des matrices : si A et B sont deux matrices réelles de même taille, la commande `A<B` est la matrice des booléens `A(i,j)<B(i,j)`.

On peut combiner ceci avec la commande `find` :

```
a= 1:2:12  //on obtient [1,3,5,7,9,11]
b=find(a>3) // positions des coefficients >3 : [3,4,5,6]
a(b) //  vecteur composé des coefficients de a qui sont >3 : [5,7,9,11]
```

4. Créer une fonction

Une fonction en mathématiques est définie par deux expressions comme dans l'exemple $f : \mathbb{R}^2 \longrightarrow \mathbb{R}, (x, y) \mapsto x^2 + y^3$. La définition d'une fonction dans scilab est tout à fait analogue, comme dans la commande suivante qui affecte à la variable "mafonction" la fonction en question :

```
> deff(´z=mafonction(x,y)´,´z=x^2+y^3´)
> mafonction(3,4)
```
On constate que pour définir une fonction, on utilise deux expressions (mises entre apostrophes), la première donnant le nom de la fonction et de ses arguments, et la deuxième donnant la formule qui la calcule. Une fois qu'on a créé une fonction, on peut y faire appel comme à n'importe quelle autre variable, comme dans les exemples suivants :
```
> deff(´z=f(x)´,´z=x^2´)
> deff(´z=eval2(g)´,´z=g(2)´)
> eval2(f)
> x=1 :2 :10 ; feval(x,f)
```
Remarquez ici l'usage de la fonction `feval` pour appliquer terme à terme une fonction à un vecteur.

5. Programmation

Dans ce qui suit, on commence par écrire un fichier texte, et on le fait ensuite prendre en compte par scilab. Pour écrire ce fichier texte, on peut, soit utiliser un éditeur de textes externe, comme emacs, ou utiliser l'éditeur de texte inclus dans scilab. Un fichier programme peut être une bibliothèque de fonctions (fichier .sci), ou un fichier exécutable (fichier .sce). La distinction entre ces deux types de fichiers, importante dans les versions 4 de scilab, est devenue non pertinente dans la version 5. Pour faire prendre en compte un fichier par scilab, on peut soit l'ouvrir dans l'éditeur de texte intégrés, puis utiliser le menu "Exécuter", soit taper la commande `exec 'nom-du-fichier.sci'`.

5.1. **Bibliothèque de fonctions.**

Une bibliothèque de fonctions est une suite de paragraphes de la forme :
```
function z=lamemefonction(x,y)
  z=x^2+y^3
endfunction
```
On peut ensuite appeler cette fonction en tapant
```
lamemefonction(2,3)
```

5.2. **Fichier exécutable.**

Voici par exemple un script qui demande à l'utilisateur de rentrer les coefficients a, b, c d'un polynôme du second degré et en trace le graphe sur l'intervalle $[0, 1]$:
```
// un script passionnant
a=input(" Rentrer la valeur de a : ");
b=input(" Rentrer la valeur de b : ");
c=input(" Rentrer la valeur de c : ");
x=[0:.01:1] ;
y=a*x.*x+b*x+c*ones(x);
plot2d(x,y)
```
Ce n'est rien d'autre qu'une suite de commandes, qu'on aurait aussi bien pu taper dans l'interpréteur de commandes. Notez ici l'usage de // pour commencer une ligne de commentaires.

5.3. **Les structures de contrôle.**

5.3.1. *La boucle for.* La syntaxe générale d'une boucle `for` est
```
for i=vecteur
  suite-d-instructions
end
```

L'indice i prend successivement la valeur de chaque coordonnée du vecteur. La fin de la liste d'instruction est marquée par le `end`. Donnons un exemple trivial :

```
s=0
for i=1:1000
  s=s+1/i^2
end
```

5.3.2. *La boucle while.* Une variante de ce qui précède est la boucle `while` (tant que), dont la syntaxe est

```
while condition
  suite-d-instructions
end
```

5.3.3. *Les instructions conditionnelles.* On utilisera essentiellement des constructions `if then else`, dont la syntaxe est la suivante

```
if condition-principale then
  suite-d-instructions
elseif
  autre-suite-d-instructions
else
  encore-des-instructions
end
```

Les conditions sont données par des opérateurs booléens. Il peut avoir un nombre quelconque (éventuellement nul) de `elseif`, mais au plus un seul `else`.

5.3.4. *Disjonction de cas.* Pour faire telle ou telle chose, suivant qu'une variable prend telle ou telle valeur :

```
select var
  case val1 then
    instructions pour le cas ou la variable var prend la valeur val1
  case val2 then
    instructions pour le cas ou la variable var prend la valeur val2
// et ainsi de suite
  else
    instructions pour les autres cas
end
```

5.3.5. *Appels récursifs.* Une fonction peut s'appeler elle même. l'exemple le plus classique est la définition de la factorielle :

```
function f=fact(n)
  if n<=1 then
    f=1
  else
    f=n*fact(n-1)
  end
endfunction
```

Ceci donne une programmation élégante, mais souvent inefficace en termes d'occupation mémoire. C'est en pratique peu utilisé pour du calcul numérique, travaillant avec des données de grande taille.

5.3.6. *Temps d'exécution.* La commande `timer()` lance et arrête un chronomètre, ce qui permet de mesurer le temps CPU utiliser par un calcul :

```
timer()
s=0; for i=1:1000000, s=s+i, end ;
temps=timer() // la variable temps contient le temps d'exécution du programme.
```

5.3.7. *Entrées-Sorties.* Le résultat d'un calcul, par exemple une grosse matrice, peut être sauvegardé dans un fichier texte, puis chargé quand on en a besoin, comme le montre l'exemple suivant :

```
> a=rand(50,50) ;
> write('truc.dat',a)
> a=zeros(1)
> a=read('truc.dat',2,2)
a =
! 0.2113249 0.4094825 !
! 0.4256872 0.2659857 !
```

Ici, la variable a contient maintenant la sous-matrice principale de taille 2x2, du résultat de `a=rand(50,50)`.

5.3.8. *Débogage.* Il faut quelques fois faire la chasse aux erreurs de programmation. Une méthode consiste à introduire aux endroits stratégiques d'une procédure des instructions `pause`. Lors de l'exécution d'une telle procédure, scilab s'arrête quand il rencontre une pause et vous avez la main pour demander à scilab d'afficher les valeurs des différentes variables à cet instant de l'éxécution du programme ; vous pouvez ensuite dire à scilab de repartir avec l'instruction `resume`. Il repartira alors jusqu'à la prochaine pause, comme si rien ne s'était passé pendant l'interruption.

On peut aussi, demander à ce que le programme affiche des résultats (les valeurs de certaines variables) au cours de son exécution, afin de suivre son déroulement. Exemple :

```
for i=1:9, disp(i),end
```

6. Graphiques

Scilab peut ouvrir plusieurs fenêtres graphiques, qui sont numérotées à partir de 0. La commande `clf()` (resp.`clf(i)`) efface le contenu de la fenêtre active, (resp. de la fenêtre `i`). Enfin `scf(i)` active la fenêtre `i` ; ensuite, tous les ordres d'affichage afficheront dans le fenêtre `i`.

6.1. **En dimension 2.** Pour dessiner une courbe dans le plan, on fait afficher un certain nombre de points $(x_i, y_i)_{i=1,n}$. A cet effet, on construit les vecteurs $x = (x_i)_{i=1,n}$ et $y = (y_i)_{i=1,n}$ de même taille, et la commande de base est

```
plot2d(x,y,[arguments optionnels])
```

Voici des exemples montrant certaines utilisations des arguments optionnels :

```
// x initialisation
x=[0:0.1:2*%pi]';
// tracé du graphe de x->sin(x) sans options
plot2d(x,sin(x))
// tracé de plusieurs graphes
clf()
plot2d(x,[sin(x) sin(2*x) sin(3*x)])
// On fixe la zone qui est représentée
clf()
```

```
plot2d(x,[sin(x) sin(2*x) sin(3*x)],rect=[0,0,6,0.5])
//
clf()
plot2d(x,[sin(x) sin(2*x) sin(3*x)],..
  style=[1,2,3],leg="L1@L2@L3",nax=[2,10,2,10],rect=[0,-2,2*%pi,2])
// vue isométrique
clf()
plot2d(x,sin(x),1,frameflag= 4)
// utilisation de coordonnées logarithmiques
y=exp(x); clf()
plot2d(x,y,logflag="nl")
```

6.2. **En dimension 3.** Pour tracer le graphe d'une fonction $f : \mathbb{R}^2 \to \mathbb{R}$, on construit deux vecteurs $x = (x_i)_{i=1,n}$ et $y = (y_i)_{i=1,p}$, puis la matrice $z = (f(x_i, y_j))_{i,j}$, et on utilise la commande :
`plot3d(x,y,z)`, ainsi que d'autres commandes plus spécialisées
Voici une session montrant quelques exemples :

```
x=linspace(0,2*%pi,10);z=cos(x)'*sin(x);
xtitle("dessin 3D");plot3d(x,x,z)
clf()
x=linspace(0,2*%pi,40);z=cos(x)'*sin(x);
xtitle("dessin 2D avec couleurs");Sgrayplot(x,x,z)
clf()
xtitle("courbes de niveau");contour2d(x,x,z,6)
clf()
x=linspace(0,2*%pi,15);fx=-sin(x)'*sin(x);fy=cos(x)'*cos(x);
xtitle("champ de gradient");champ(x,x,fx,fy)
```

6.3. **Gestion de la fenêtre graphique.** On peut avoir envie d'afficher plusieurs graphes simultanément

6.3.1. *Dans plusieurs fenêtres.*

```
x=1:.01:10;y=sin(x);z=.3*sin(5*x);
scf(0);clf();plot2d(x,y);xtitle("graphe de x->sin(x)")
scf(1);clf();plot2d(x,z);xtitle("graphe de x->3sin(5x)")
scf(2);clf();plot2d(x,y+z);xtitle("graphe de x->sin(x)+3sin(5x)")
```

6.3.2. *En subdivisant une fenêtre.*

```
x=linspace(0,2*%pi,40);z=cos(x)'*sin(x);
subplot(2,2,1); xtitle("dessin 2D avec couleurs");Sgrayplot(x,x,z)
subplot(2,2,2);xtitle("courbes de niveau");contour2d(x,x,z,6)
x=linspace(0,2*%pi,15);fx=-sin(x)'*sin(x);fy=cos(x)'*cos(x);
subplot(2,2,3);xtitle("champ de gradient");champ(x,x,fx,fy)
x=linspace(0,2*%pi,10);z=cos(x)'*sin(x);
subplot(2,2,4);xtitle("dessin 3D");plot3d(x,x,z)
```