Révisions : Programmation en Scilab

Table des matières

Scilab est un logiciel de calcul scientifique gratuit, qui permet d'effectuer tous les calculs d'une calculatrice, voire plus. Son langage est particulièrement adapté au traitement de matrices, tableaux, vecteurs, etc... Il possède de nombreuses fonctions intégrées, de calcul, traitement, représentation graphique, et simulation d'événements aléatoires.

Scilab n'est pas en revanche un logiciel de calcul formel, et ne permet pas, par exemple, de calculer la dérivée d'une fonction.

Il est disponible gratuitement à l'adresse `http://www.scilab.org`.

1 Mise en route

1.1 Ouverture d'une session Scilab

Pour démarrer une session Scilab, il suffit de lancer l'application, par exemple en double-cliquant sur son icône sur le bureau. Vous êtes alors accueillis par plusieurs fenêtres (parfois une seule...). Celle qui nous intéresse est la *console*, qui affiche, entre autre, un symbole, `-->`. Ce symbole indique que Scilab attend que vous entriez une commande.

1.2 Premiers pas

La ligne de commande dans la console permet de taper des opérations élémentaires, comme pour une calculatrice scientifique. Il faut taper tous les opérateurs, et bien mettre des parenthèses autour de la formule. Une fois la formule correctement entrée, appuyer sur la touche "entrée" affiche le résultat. Par exemple :

```
--> 1+1
 ans =

   2.
```

`ans` est l'abréviation de *answer*, il s'agit du nom de la variable où la réponse est stockée.

2 Variables

Scilab permet de stocker, manipuler et récupérer des valeurs. Ces valeurs sont stockées dans des *variables*, désignées par un *nom*, et qui possèdent un *contenu* (qui renfermera ainsi la valeur souhaitée). Le nom d'une variable doit toujours commencer par une lettre, et peut contenir des chiffres, des lettres et le symbole `_`, mais **pas d'espace, caractères accentués ou autre symbole**.

La commande `x=9` permet à Scilab de créer une variable de nom `x` (si elle n'existe pas déjà), et de stocker dans celle-ci la valeur 9. L'opérateur d'affectation est ainsi `=`, et la formule générale est

```
--> nom = contenu
```

Les variables peuvent contenir divers types de valeurs : des entiers, des réels, des chaînes de caractères, des vecteurs, des matrices, etc....

Remarque 2.1

Il est parfois utile de vider la mémoire de l'ordinateur. Pour ce faire, il suffit d'utiliser la commande `clear`. Ainsi,

- `clear x` permet de supprimer la variable `x`,
- `clear` permet de supprimer toutes les variables et fonctions.

Nous avons déjà vu la variable `ans`, qui contient la réponse du dernier calcul. Scilab possède d'autres variables déjà prédéfinies :

variable	contenu	variable	contenu
`ans`	dernière réponse	`%inf`	presque infini
`%e`	la constante d'Euler $e \approx 2.718218\ldots$	`%pi`	$\approx 3.141592\ldots$
`%t`	constante booléenne VRAI	`%f`	constante booléenne FAUX

On peut effectuer plusieurs commandes sur une même ligne : pour cela on sépare deux commandes successives par un `;` si on ne désire pas que le résultat s'affiche à l'écran (lorsqu'on effectue un calcul intermédiaire par exemple). Si au contraire on veut voir le résultat affiché, il faut séparer les deux commandes par un `,`

```
--> a=2; b=2;3*a+8*b^3 // le résultat de ce dernier calcul sera affiché
 ans =

   70.
```

On rappelle que `//` permet de commenter une commande.

3 Programmation d'algorithmes

3.1 Les opérateurs de comparaison

Comparer des nombres ou vérifier si une affirmation est vraie ou fausse sont des tests utiles. Voici les commandes correspondantes :

Égal	Différent	Inférieur	Supérieur	Inférieur ou égal	Supérieur ou égal
==	<>	<	>	<=	>=

Vrai	Faux	Et	Ou
`%t`	`%f`	`&`	|

Lorsque l'on veut comparer deux vecteurs, les tests `==` et `<>` comparent terme à terme.

```
--> v=[1,2,5]; w=[5,3,5]; v==w
 ans  =

  F F T
```

3.2 Les boucles

On distingue essentiellement deux types de boucles :

- *La boucle de type* `for`
  La structure de boucle la plus simple pour un nombre connu d'itérations s'écrit avec **for ... end** qui signifie "Pour ... fin de pour".

```
--> x=1; for i=1:7 x=x*i; end ; x // factorielle de 7
 x  =

    5040.
```

- *La boucle de type* `while`
  Si l'on veut que la boucle s'arrête lorsqu'un objectif donné est atteint, on utilisera la forme **while ... end** qui signifie "Tant que ... fin de tant que".

```
--> x=1; while x<4 x=3*x, end
 x  =

    3.
 x  =

    9.
```

3.3 Les tests

Le test *if-then-else* est incorporé en Scilab. Les structures classiques sont les suivantes :

- **if ... then ... else ... end** ("Si... alors... sinon... fin de si")
- **if ... then ... elseif ... then ... else ... end** ('Si... alors... ou si... alors... sinon... fin de si")

```
--> x=-1; if x < 0 then y=-x, else y=x, end
 y  =

    1.
```

4 Construction de vecteurs et de matrices numériques

4.1 Vecteurs

4.1.1 Création d'un vecteur en donnant la liste des éléments

Pour définir un vecteur de petite taille, on donne la liste des éléments du vecteur entre crochets, séparés par des virgules (ou des espaces) pour un vecteur ligne et séparés par des points-virgules pour un vecteur colonne. Par exemple :

```
--> v=[2,-3*%pi,7] // vecteur ligne
 v  =

    2.  - 9.424778    7.

--> w=[2;-3*%pi;7] // vecteur colonne
 w  =

    2.
  - 9.424778
    7.
```

La commande ' permet de basculer d'un vecteur ligne à un vecteur colonne (et réciproquement).

```
--> w=[2,-3*%pi,7]' // vecteur transposé
 w  =

    2.
  - 9.424778
    7.
```

4.1.2 Création d'un vecteur dont les éléments forment une suite arithmétique

Pour un vecteur ligne `v` dont les éléments forment une suite arithmétique de premier terme `a`, de raison `r` et ne dépassant pas `b`, on utilise la commande : `v=a:r:b`. Par exemple :

```
--> v=-3:2:14 // vecteur à incrément constant
 v  =

  - 3.  - 1.    1.    3.    5.    7.    9.    11.    13.
```

Remarque 4.1

Lorsque le pas est égal à 1, on peut ne pas le mentionner et au lieu d'écrire `v=-2:1:2`, on peut simplement écrire `v=-2:2`.

4.1.3 Création d'un vecteur avec linspace

Pour un vecteur ligne de pas constant (pas forcément déterminé à l'avance mais que Scilab calcule), dont le nombre de termes, le premier terme et le dernier terme sont imposés, on utilise la commande `v=linspace(début, fin, nombre de valeurs)`.

```
--> v=linspace(-5,8,6)
 v  =

  - 5.  - 2.4    0.2    2.8    5.4    8.
```

4.2 Matrices

4.2.1 Création d'une matrice de petite taille

Pour définir une matrice de petite taille, on donne la liste des éléments de la matrice entre crochets, les éléments de chaque ligne étant séparés par des virgules (ou des espaces) et les lignes étant séparées par des points-virgules. Par exemple :

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

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

4.2.2 Création d'une matrice de grande taille

Pour une matrice de grande taille, on peut lorsque c'est possible, définir chaque ligne comme on définit un vecteur, les lignes étant toujours séparées par des points-virgules.

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

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

Dans toute la suite, nous parlerons de matrices, y compris pour désigner des vecteurs lignes (matrices de format (1,n)) ou colonnes (matrices de format (n,1)).

4.2.3 Matrices prédéfinies

commande	contenu
`ones(n,m)`	matrice à n lignes et m colonnes dont tous les éléments sont égaux à 1.
`zeros(n,m)`	matrice à n lignes et m colonnes dont tous les éléments sont égaux à 0.
`eye(n,m)`	matrice à n lignes et m colonnes avec des 1 sur la diagonale et des 0 ailleurs.

```
--> A=eye(2,3)
 A  =

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

Méthode 4.2

Il est souvent plus rapide de créer une matrice `A` de $\mathcal{M}_{n,m}(\mathbb{R})$ après l'avoir initialisée, avec `A=zeros(n,m)` puis de la modifier, plutôt que de devoir la créer élément par élément sans avoir au préalable défini son format en initialisant.

```
--> A=zeros(5,5); for i=1:5 A(i,i)=i; end; A
 A  =

    1.    0.    0.    0.    0.
    0.    2.    0.    0.    0.
    0.    0.    3.    0.    0.
    0.    0.    0.    4.    0.
    0.    0.    0.    0.    5.
```

5 Opérations sur les matrices

5.1 Concaténation

La concaténation permet de juxtaposer vecteurs et matrices dont les formats sont compatibles. Par exemple :

```
--> u=1:3; v=[u,u,u]
 v  =

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

Attention à ne pas confondre virgule et point-virgule.

```
--> u=1:3; v=[u;u;u]
 v  =

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

5.2 Opérations arithmétiques classiques

On considère deux matrices A et B, pour lesquelles les opérations ci-dessous sont possibles, un réel k et un entier n.

Syntaxe	`k*A`	`A+B`	`A-B`	`A*B`	`A^n`	`inv(A) ou A^(-1)`	`A'`
Signification	kA	$A+B$	$A-B$	AB	A^n	A^{-1}	${}^tA$

```
--> A=[1,0,0;0,2,0;0,0,3]; B=[1,1,1;2,2,2;3,3,3]; A+B, A*B
 ans  =

    2.    1.    1.
    2.    4.    2.
    3.    3.    6.
 ans  =

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

5.3 Opérations arithmétiques pointées

Ce sont des opérations élément à élément. On considère deux matrices $A = (a_{i,j})$ et $B = (b_{i,j})$ de même format et un entier n.

Syntaxe	`A.* B`	`A./B`	`A.^n`
Matrice renvoyée	$(a_{i,j}b_{i,j})$	$(a_{i,j}/b_{i,j})$	$(a_{i,j}^n)$

L'opération `A./B` n'est licite que si les coefficients de B sont tous non nuls.

```
--> A=[1,0,0;0,2,0;0,0,3]; B=[1,1,1;2,2,2;3,3,3]; A.*B, A./B
 ans  =

    1.    0.    0.
    0.    4.    0.
    0.    0.    9.
 ans  =

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

5.4 Fonctions matricielles

Si A est une matrice et si f est une fonction connue de Scilab (voir la section 7.1), la commande `f(A)` retourne la matrice dont les éléments sont les images par f des éléments de A. Par exemple si $f(x) = \sqrt{x+5}$:

```
--> A=[1,-2,7;4,2,0]; B=sqrt(A+5),
 B  =

    2.4494897    1.7320508    3.4641016
    3.           2.6457513    2.236068
```

Scilab a donc ajouté 5 à chaque élément de la matrice A (en fait `A+5` est un raccourci pour `A+5*ones(2,3)`), puis a pris la racine carrée de chacun des nombres obtenus.

6 Manipulation des éléments d'une matrice

6.1 Changement d'éléments

Si la matrice A a déjà été créée, on peut changer certains éléments en donnant à chacun de ces éléments une nouvelle valeur. Par exemple si on a déjà défini la matrice $A = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -4 \end{pmatrix}$, alors

```
--> A(1,2)=5; A(2,2)=1
 A  =

    0.    5.    0.
    0.    1.  - 4.
```

6.2 Recherche d'éléments dans une matrice

Si A est une matrice, la commande `A(a:r:b)` retourne les éléments de A dont les positions sont a, $a+r$, $a+2r$, ..., sans dépasser b. Les positions sont données en commençant par la première colonne de haut en bas, puis la deuxième, etc....

```
--> A=[1,-2,7;4,2,0;1,1,4], B=A(1:2:9),
 A  =

    1.  - 2.    7.
    4.    2.    0.
    1.    1.    4.
 B  =

    1.
    1.
    2.
    7.
    4.
```

6.3 Recherche conditionnelle d'éléments dans une matrice : la fonction find

La fonction `find` permet de trouver les éléments d'une matrice ayant une certaine propriété. Tout comme avant, les positions sont données en commençant par la première colonne de haut en bas, puis la deuxième, etc.... La fonction `find` retourne la position de chacun des éléments trouvés.

```
--> A=[1,-2,7;4,2,0;1,1,4], x=find(A<2 & A>0)
 A  =

    1.  - 2.    7.
    4.    2.    0.
    1.    1.    4.
 x  =

    1.    3.    6.
```

7 Fonctions usuelles prédéfinies

7.1 Pour l'analyse

- `sqrt(x)` retourne la racine carrée de x pour x réel positif ou nul.
- `log(x)` retourne le logarithme de x pour x réel strictement positif.

- `exp(x)` retourne l'exponentielle du réel x.
- `abs(x)` retourne la valeur absolue du réel x.
- `floor(x)` retourne la partie entière du réel x.

7.2 Pour simuler des lois usuelles

- `rand` retourne un nombre réel pris aléatoirement entre 0 et 1.
- `rand(n,r)` avec n et p entiers positifs, retourne une matrice $\mathcal{M}_{n,r}(\mathbb{R})$ de nombres pris aléatoirement entre 0 et 1.

```
--> rand(2,3)
 ans  =

    0.0002211    0.6653811    0.8497452
    0.3303271    0.6283918    0.6857310
```

- La fonction `grand` permet de simuler toutes les lois usuelles (binomiale, Poisson, ...) du cours de probabilités. Si la loi simulée dépend de m paramètres, elle s'utilise toujours sous la forme

  `grand(n,r,'loi', paramètre 1, paramètre 2, ..., paramètre m)`

  Cette commande renvoie une matrice de $\mathcal{M}_{n,r}(\mathbb{R})$ dont les éléments sont les valeurs prises par nr variables aléatoires indépendantes qui suivent la loi en question.

7.2.1 Simulation de variables aléatoires discrètes

- Loi uniforme discrète sur $[\![a, b]\!]$ avec $a, b \in \mathbb{N}$ tels que $a < b$:

  `grand(n,r,'uin',a,b)`

```
--> grand(1,3,'uin',3,10)
 ans  =

    7.    8.    4.
```

- Loi binomiale de paramètres N et p :

  `grand(n,r,'bin',N,p)`

- Loi géométrique de paramètre p :

  `grand(n,r,'geom',p)`

- Loi de Poisson de paramètre λ :

  `grand(n,r,'poi',lambda)`

7.2.2 Simulation de variables aléatoires à densité

- Loi uniforme continue sur $[a, b]$ avec $a, b \in \mathbb{R}$ tels que $a < b$:

  `grand(n,r,'unf',a,b)`

```
--> grand(1,5,'unf',1,%pi)
 ans  =

    1.2408532    2.9611325    2.3701125    2.6965854    2.8812406
```

- Loi exponentielle de paramètre λ :

  `grand(n,r,'exp',1/lambda)`

- Loi normale $\mathcal{N}(\mu, \sigma^2)$:

  `grand(n,r,'nor',mu,sigma)`

7.3 Pour les statistiques

- `mean(A)` retourne la moyenne des éléments de la matrice A.

```
--> A=[1,-2,7;4,2,0;1,1,4]; mean(A)
 ans  =

    2.
```

- `stdev(A)` retourne l'écart type des éléments de la matrice A.

```
--> A=[1,-2,7;4,2,0;1,1,4]; stdev(A)
 ans  =

    2.6457513
```

- `median(A)` retourne la médiane des éléments de la matrice A.

```
--> A=[1,-2,7;4,2,0;1,1,4]; median(A)
 ans  =

    1.
```

7.4 Pour les matrices

- `min(A)` retourne le plus petit élément de la matrice A.
- `max(A)` retourne le plus grand élément de la matrice A.
- `sum(A)` retourne la somme des éléments de la matrice A.

```
--> u=1:10, s=sum(u),
 u  =

    1.    2.    3.    4.    5.    6.    7.    8.    9.    10.
 s  =

    55.
```

- `cumsum(A)` retourne une matrice de même format que A dont les éléments sont les sommes partielles des éléments de A, ces éléments étant ajoutés en commençant par ceux de la première colonne, puis ceux de la deuxième colonne, etc...

```
--> u=1:10, t=cumsum(u),
 u  =

    1.    2.    3.    4.    5.    6.    7.    8.    9.    10.
 t  =

    1.    3.    6.    10.    15.    21.    28.    36.    45.    55.
```

- `A'` retourne la transposée de la matrice A.
- `rank(A)` retourne le rang de la matrice A (nous verrons cette notion dans le cours sur les espaces vectoriels).
- `inv(A)` retourne l'inverse de la matrice A si celle-ci est inversible.
- `size(A)` retourne un vecteur ligne égale à `[n p]`, si A est une matrice de $\mathcal{M}_{n,p}(\mathbb{R})$.

```
--> A=[1:4;2:5;3:6]; size(A) // la matrice possède 3 lignes et 4 colonnes.
 ans  =

    3.    4.
```

- `spec(A)` retourne le spectre de la matrice A, c'est-à-dire l'ensemble de ses valeurs propres. Cette notion sera abordée dans le chapitre sur la diagonalisation des matrices.

```
--> A=[2 5;0 6]; spec(A),
 ans  =

    2.
    6.
```

7.5 Les fonctions "retour de résultats"

7.5.1 La fonction input

La fonction `input` permet à l'utilisateur de rentrer une valeur. la syntaxe est : **x=input('message')**.

- 'message' est une chaîne de caractères qui s'affiche dans la fenêtre de commandes.
- x est une variable où on va stocker la valeur entrée par l'utilisateur en réponse à cette question.

```
--> x=input('Quelle valeur voulez-vous affecter à la variable x ?')
Quelle valeur voulez-vous affecter à la variable x ?2
 x  =

    2.
```

7.5.2 La fonction disp

La fonction `disp` permet l'affichage du contenu de variables de toute nature (nombres, matrices, texte). Cette fonction est nécessaire pour afficher vos résultats lorsque vous utilisez *SciNotes*.

```
--> a=2; A=[2 1;3 5]; B=a*A; disp('La matrice est:'), disp(B)

 La matrice est:

    4.    2.
    6.    10.
```