Fiche récapitulative commandes informatique

Cette fiche donne la liste de toutes les commandes Scilab qui doivent être connues par les candidats (d'après les programmes officiels de première et deuxième année).

On rappelle que `%pi` donne une approximation de π, que `%e` donne une approximation de $e = \exp(1)$, que l'entrée des données se fait avec la commande `input('....')` et la sortie des données (affichage) se fait avec la commande `disp()`.

1 Les structures conditionnelles (if) et itératives (boucles)

a) Les structures conditionnelles

• S'il n'y a que 2 cas :
- Si l'on ne fait des instructions que dans un des deux cas :
  `if..(test)...then..(instructions)...end`
- Si l'on fait des instructions dans chacun des deux cas :
  `if ...(test)...then..(instructions1)...else..(instructions2)...end`

• S'il y a trois cas ou plus : (dans le cas particulier de l'exemple ci-dessous, il y a 4 cas)

```
if ..(test1)....then..(instructions1)...elseif ..(test2).. then..(instructions2).. elseif..(test3)..
then ..(instructions3)..else ...(instructions4)..end
```

b) Les structures itératives (boucles)

• Si l'on sait d'avance le nombre de répétitions souhaitées :

```
for (k=**:**) do...(instructions)..end
```

• Si Scilab doit tester à chaque boucle si on continue ou non les répétitions :

```
while..(test)...do..(instructions)...end
```

2 Affectation et tests d'hypothèses

a) Affectation

L'instruction `x=y` affecte la valeur de `y` à la variable (case mémoire) `x`.

b) Tests d'hypothèse

Les instructions suivantes, appelées test d'hypothèses (ou comparaisons) ne font que renvoyer "vrai" ou "faux". Ce sont les seules instructions que l'on peut placer dans la partie `(test)` des structures itératives et conditionnelles (cf partie 1.) :
• Test d'une seule condition :
Soient `x` et `y` deux variables réelles.

- L'instruction `x==y` renvoit "vrai" si `x` et `y` prennent les mêmes valeurs et "faux" sinon .
  Attention, il faut bien taper `x==y` *dans un test et non* `x=y` *qui est une affectation. Attention à ne pas tomber dans ce piège aux concours !*
- L'instruction `x<>y` renvoit "vrai" si `x` et `y` prennent des valeurs différentes et "faux" sinon .
- Les seules autres comparaisons possibles sont `x<y`, `x<=y` , `x>y` et `x>=y`.

• soient `x`, `y` et `z` des variables réelles.
Si l'on veut tester deux conditions (un encadrement par exemple : $x < y < z$), on ne peut pas taper `x < y < z` ! Il faut utiliser les commandes : `&` (et) et `|` (ou) :
Par exemples,

— L'instruction `x < y & y<z` renvoit "vrai" si $x < y < z$ et "faux" sinon.
— L'instruction `x == y | y==z` renvoit "vrai" si on a soit $x = y$, soit $y = z$, soit les 2 (le "ou" mathématiques est toujours inclusif : l'un ou l'autre ou les deux).

• Les tests sont aussi possibles sur des matrices.
En effet, par exemple, si A et B sont deux matrices de même taille, l'instruction `A==B` renvoit une matrice dont le (i,j)-ème coefficient vaut "vrai" si $a_{i,j} = b_{i,j}$ et "faux" sinon.
Aussi, si x est un vecteur ligne, l'instruction `x==5` renvoit un vecteur ligne dont le i-ème coefficient vaut "vrai" (codé par 1) si $x_i = 5$ et "faux" (codé par 0) sinon.
Deux fonctions sont à connaître sur les tests :
— `sum` : Par exemple, l'instruction `sum(x==5)` renvoit le nombre de coefficients de x valant 5.
— `find` : Par exemple, l'instruction `find(x==5)` renvoit la liste des rangs des coefficients de x valant 5.
Exemple :

```
--> x=[ 1 9 8 6 4 3 3 9 7 3 8 5 3 3];

-->sum(x==3)
 ans  =

    5.

-->find(x==3)
 ans  =

    6.    7.    10.    13.    14.
```

3 Calcul matriciel

a) Définition d'une matrice

• Une matrice est définie en crochets []. On indique un changement de colonne par une virgule et un changement de ligne par un point-virgule.
— Vecteur ligne : `[ , , ,...,]`
— Vecteur colonne : `[ ; ; ;...;]`
— Matrices (n, p) : `[ ,...,  ;.........;  ,...,]`

• Les matrices prédéfinies en Scilab :

`zeros(n,p)`	matrice de taille (n, p) dont tous les coefficients sont nuls
`ones(n,p)`	matrice de taille (n, p) dont tous les coefficients valent tous 1
`eye(n,n)`	matrice identité d'ordre n (de taille (n, n))
`diag([a1,a2,...,an])`	matrice diagonale d'ordre n dont les coefficients diagonaux valent $a1$, $a2$, $...an$
`a:h:b`	vecteur ligne des valeurs comprises entre a et b avec un pas h : $(a \quad a+h \quad a+2h \quad ... \quad b)$
`a:b`	si on omet le pas, il est considéré comme valant $h = 1$.
`linspace(a,b,n)`	vecteur ligne contenant exactement n valeurs régulièrement espacées entre a et b

• *Remarque : on peut définir une matrice par bloc à l'aide de cette matrice usuelle. Par exemple, la matrice* $A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$ *peut être définie par l'instruction :*

```
A=[[ones(4,1);0],zeros(5,3),[ones(4,1);0]]
```

b) Opérations sur les matrices

Soient A et B deux matrices.
• Il faut juste retenir qu'une opération sans "point", du type `A *B`, effectue une opération matricielle (le produit matriciel $C = AB$ dans cet exemple tel que $c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}$), alors qu'une opération "pointée", du type `A.*B`, effectue l'opération coefficient par coefficient (elle renverrait la matrice C telle que $c_{i,j} = a_{i,j} \times b_{i,j}$ dans cet exemple).

Bien sûr, ces opérations ne sont effectuées que si les tailles des matrices le permettent, sans quoi Scilab renvoit un message d'erreur.

• Soit f une fonction définie préalablement (ou prédéfinie en Scilab). L'instruction `f(A)` applique toujours la fonction f coefficient par coefficient, i.e., elle renvoit donc la matrice C telle que $c_{i,j} = f(a_{i,j})$.
Fonctions exigibles : `log(A)`, `exp(A)`, `sqrt(A)`, `abs(A)`, `floor(A)`.

• Extraction d'un coefficient ou d'une sous-matrice :
Soit A une matrice :

`A(2,3)`	Coefficient de la ligne 2, colonne 3 de A
`A(:,3)`	3 ème colonne de A
`A(1,:)`	1 ère ligne de A
`A(:, [2:4])`	sous-matrice des colonnes 2à 4 de A

c) Fonctions matricielles

commande	définition
`r=rank(A)`	r = rang de A (dimension de l'espace vectoriel image)
`[n p]=size(A)`	n = nombre de lignes de A, p =nombre de colonnes de A
`l=length(A)`	$l = n \times p$ (nombre total de coefficients)
`B=A'`	B = transposée de A
`B=inv(A)`	$B = A^{-1}$ (à condition que A soit inversible bien sûr)
`m=min(A)`	m = le plus petit des coefficients de A
`M=max(A)`	M = le plus grand des coefficients de A
`s=sum(A)`	s = la somme des coefficients de A
`m=mean(A)`	s = la moyenne des coefficients de A (idem à `m=sum(A)/length(A)`)
`S=cumsum(x)`	(*n'existe que si x est un vecteur ligne ou vecteur colonne*) S = matrice de même taille que x contenant les sommes partielles des éléments de x.
`u=spec(A)`	u = la liste des valeurs propres de A. En revanche, si on tape :
`[P,D]=spec(A)`	renvoit des matrices P inversible et D diagonale telles que $A = PDP^{-1}$ (*si A est diagonalisable ! Sinon, message d'erreur.*)
`m=min(A,B)`	(A et B doivent être de même taille (n,p)) m = matrice (n,p) t.q. $\forall i,j,\ m_{i,j} = min(a_{i,j}, b_{i,j})$
`M=max(A,B)`	(A et B doivent être de même taille (n,p)) M = matrice (n,p) t.q. $\forall i,j,\ M_{i,j} = max(a_{i,j}, b_{i,j})$

4 Etude de suites et de fonctions

• Soit u un vecteur ligne contenant les termes successifs d'une suite (u_n). Pour représenter graphiquement la suite (u_n), on tape : `plot(u,'+')` (dans ce cas, Scilab considère, par défaut, que le rang de chaque terme est le numéro du terme dans le vecteur, le premier terme est donc u_1, le second u_2, ...). Si le rang commence à 0 et finit à $n-1$ par exemple, on tape `plot(0:n-1,u,'+')`.

• Soient x et y deux listes de valeurs. Si on souhaite tracer la courbe approchée de y en fonction de x (i.e. les points de coordonnées (x_i, y_i) reliés par des segments), on tape `plot2d(x,y)`.

• Soient x, y et z deux listes de valeurs. Si on souhaite tracer la surface approchée de z en fonction de (x, y), on tape `plot3d(x,y,z)`.

• On rappelle que les fonctions à variable réelle prédéfinies en Scilab exigibles du programme sont : `log`, `exp`, `sqrt`, `abs`, `floor`.

• Pour définir une nouvelle fonction, la structure est la suivante :

```
function sorties=nom_de_la_fonction(entrées)
    instructions
endfunction
```

— Pour une fonction f de $\mathbb{R}$ dans $\mathbb{R}$, ça donne :

```
function y=f(x)
    ........
    y=........
endfunction
```

On peut aussi définir plus rapidement une fonction simple d'une variable par l'instruction :
`deff('y=f(x)', 'y=....')`

Pour tracer la courbe de f sur le segment $[a, b]$, on tape alors `-->fplot2d(a:h:b,f)`.

— Pour une fonction f de 2 variables (de $\mathbb{R}^2$ dans $\mathbb{R}$) , ça donne :

```
function z=f(x,y)
    ........
    z=........
endfunction
```

On peut aussi définir plus rapidement une fonction simple de 2 variables par l'instruction :
`deff('z=f(x,y)', 'z=....')`

Pour tracer la courbe de f sur le pavé $[a, b] \times [c, d]$, on tape alors `-->fplot3d(a:h1:b, c:h2:d,f)`.

5 Simulations d'expériences aléatoires

a) La commande rand

• La commande `rand(m,n)` crée une matrice de taille (m, n) dont chaque coefficient est choisi au hasard dans le segment $[0, 1]$ (i.e. chaque coefficient est une réalisation de la loi uniforme sur $[0, 1]$).

• La commande `rand()` signifie, par défaut, `rand(1,1)` : elle effectue une seule réalisation de la loi uniforme sur $[0, 1]$.
On rappelle que pour créer un évènement de probabilité p, on utilise `rand()<p` . Par exemple :

```
if rand()<p then x=1 else x=0 end
```

affecte 1 à x avec probabilité p : ce programme effectue une réalisation d'une Bernoulli de paramètre p.

• On rappelle que l'instruction `floor(rand()*n)` choisit un entier au hasard dans l'ensemble $[\![0, n-1]\!]$.
Ainsi, l'instruction `floor(rand()*n)+1` choisit un entier au hasard dans l'ensemble $[\![1, n]\!]$ i.e. effectue une réalisation de la loi uniforme sur $[\![1, n]\!]$.

b) Lois discrètes usuelles prédéfinies par grand

— L'instruction `grand(m,n,'uin',a,b)` crée une matrice de taille (m, n) dont chaque coefficient est une réalisation de (d'une variable suivant) la loi uniforme sur $[\![a, b]\!]$ (a et b étant des entiers ici !) .
— L'instruction `grand(m,n,'bin',k,p)` crée une matrice de taille (m, n) dont chaque coefficient est une réalisation de la loi binomiale de taille k et de paramètre p.
— L'instruction `grand(m,n,'geom',p)` crée une matrice de taille (m, n) dont chaque coefficient est une réalisation de la loi géométrique de paramètre p.
— L'instruction `grand(m,n,'poi',lambda)` crée une matrice de taille (m, n) dont chaque coefficient est une réalisation de la loi de Poisson de paramètre *lambda*.

c) Lois à densité usuelles prédéfinies par grand

— L'instruction `grand(m,n,'unf',a,b)` crée une matrice de taille (m, n) dont chaque coefficient est une réalisation de la loi uniforme sur $[a, b]$.
— L'instruction `grand(m,n,'nor',mu,sigma)` crée une matrice de taille (m, n) dont chaque coefficient est une réalisation de la loi normale d'espérance mu et de variance $v = sigma^2$.
Attention : le second paramètre de grand 'nor' est l'écart-type et non la variance.
— L'instruction `grand(m,n,'exp',1/lambda)` crée une matrice de taille (m, n) dont chaque coefficient est une réalisation de la loi exponentielle de paramètre *lambda*.
Attention : il y a un gros piège (uniquement pour la loi exponentielle) : il faut donner l'inverse du paramètre dans grand 'exp'.

d) La fonction cdfnor

"cdf" est l'abréviation de "cumulative distribution fonction" i.e. fonction de répartition

• **Fonction `cdfnor` avec l'option `'PQ'` :**
L'instruction `cdfnor('PQ',X, Mean, Std)` () renvoit, si on lui donne X, Mean et Std, la valeur de $F(X)$ où F est la fonction de répartition d'une loi normale d'espérance Mean et de variance Std^2.

Remarque : L'option s'appelle `'PQ'` car on peut forcer la fonction à renvoyer 2 valeurs : la première, notée P est la valeur renvoyée ci-dessus ($P(X \leq x)$) , la seconde Q vaut $1 - P = 1 - P(X \leq x) = P(X \geq x)$.

• **Fonction `cdfnor` avec l'option `'X'` :**
L'instruction `cdfnor('X',Mean,Std,P,Q)` renvoit, si on lui donne Mean, Std, P et $Q = 1 - P$, la valeur de $X = F^{-1}(P)$ où F est la fonction de répartition d'une li normale d'espérance Mean et de variance Std^2.

• De même, l'instruction `cdfnor('Mean', Std,P,Q,X)` renvoit l'espérance de la loi normale si on lui donne l'écart type Std ainsi que les valeurs P, Q et X indiquant que pour cette loi normale on doit avoir $F(X) = P$ et $1 - F(X) = Q$ où F est sa fonction de répartition.
Enfin, l'instruction `cdfnor('Std',P,Q,X, Mean)` renvoit l'écart-type de la loi normale si on lui donne l'espérance Mean ainsi que les valeurs P, Q et X indiquant que pour cette loi normale on doit avoir $F(X) = P$ et $1 - F(X) = Q$ où F est sa fonction de répartition.

e) Statistiques univariées

a) Premier cas : étude de données brutes : Soit `U` un vecteur contenant la liste des réalisations $(x_1, ..., x_n)$ d'un n-échantillon de la loi d'une variable X.

— Une valeur approchées d'une probabilité $P(A)$ est la fréquence de l'évènement A. Par exemple, une valeur approchée de $P(X = 3)$ est `f=sum(U==3)/length(U)`, une valeur approchée de $F(t) = P(X \leq t)$ est `F=sum(U <=t)/length(U)`
— Une valeur approchée de l'espérance (moyenne théorique) de X est la moyenne empirique `m=mean(U)` de l'échantillon.
— Une valeur approchée de la variance de X est la variance empirique `v=mean((U-mean(U)).^2) =mean(U.^2)-mean(U)^2` de l'échantillon.
— Une valeur approchée de l'écart-type de X est l'écart-type empirique `sigma= sqrt(mean(U.^2)-mean(U)^2)` de l'échantillon.
(*Remarque : Il peut aussi être calculé directement par l'instruction* `st_deviation(U)` *(écart-type se dit standard deviation en anglais) mais nous éviterons cette commande*).
— La médiane de l'échantillon se calcule par l'instruction `median(U)`

b) Second cas : tabulation et étude de données tabulées :
• Cas où X est une variable discrète :
L'instruction `M=tabul(U,'i')` renvoit une matrice `M` à 2 colonnes telle que :
- La première colonne `M(:,1)` contient les différentes valeurs prises par l'échantillon `x` (modalités) rangées dans l'ordre croissant ;
- La seconde colonne `M(:,2)` contient les effectifs correspondant à chacune des modalités (nombre de réalisations de l'échantillon ayant pris cette valeur).

Remarque : Il faut aussi reconnaître le programme qui tabule les valeurs de x à la main, sans utiliser la commande `tabul` :
Supposons que l'on sait que la variable prend ses valeurs dans l'ensemble $[\![1; k]\!]$ (modalités). Le programme serait le suivant :

```
n=zeros(1,k)
for i=1:length(U) do
    n(U(i))=n(U(i))+1
end
```

Après exécution, le vecteur `n` contient la liste des effectifs de chaque modalité.

Une fois les données tabulées, on peut :
— Construire le diagramme en bâton des fréquences par les instructions :
```
x=M(:,1)
f=M(:,2)/sum(M(:,2))
 bar(x, f)
```
ou le diagramme des fréquences cumulées :
```
x=M(:,1)
f=M(:,2)/sum(M(:,2))
F=cumsum(f)
bar(x, F)
```

— Lire sur ces diagrammes la médiane, les quartiles, les déciles de l'échantillon (modalités correspondants respectivement à 50% des effectifs, à 25%, 50% et 75% des effectifs, puis à 10%, 20%.....des effectifs).
— Calculer la moyenne, la variance empirique, avec les données tabulées :
`m=sum(x.*f)`, `v=sum((x.^2).*f)-sum(x.*f)^2`

• Cas où X est une variable à densité :

— Si on veut juste tracer l'histogramme des fréquences de l'échantillon, le programme est très simple car, contrairement à l'instruction `bar` qui nécessite que les données soit déjà tabulées, l'instruction `histplot` tabule elle-même les données à condition de lui donner soit la liste des classes modales souhaitées, soit le nombre de classes souhaitées :
`histplot(0:0.01:10, U)`
ou
`histplot(20,U)`
— Si on souhaite récupérer les fréquences de chaque classe, pour faire des calculs supplémentaires, on tape (`c` désigne la liste des classes souhaitée) :
`[ind,occ,non]= dsearch(U,c)`
La sortie `non` renvoit le nombre de caractères qui n'appartiennent à aucune classe de c (permet juste de savoir si on a choisi les bonnes classes : s'il y a trop de valeurs qui n'appartiennent à aucune classe, il y a un problème !)
La sortie `occ` (la plus importante) renvoit la ,liste des effectifs associés à chaque classe de c (elle classe donc les données dans la classe c choisie).
La sortie `ind` (non utilisée en pratique) renvoit la liste des numéros de la classe à laquelle appartiennent les éléments de la série x.

Ainsi, l'instruction `f=occ/sum(occ)` donne la liste des fréquences de chaque classe modale.

f) Statistiques bivariées

Soient `x` et `y` deux vecteurs contenant un n-échantillon d'un couple (X, Y) de variables aléatoires.
• Pour calculer la covariance empirique du couple, on tape : `c= mean((x-mean(x)).*(y-mean(y)))` ou `c= mean(x.*y)-mean(x)*mean(y)`
• Ainsi, pour calculer le coefficient de corrélation linéaire du couple, on tape :

```
c= mean(x.*y)-mean(x)*mean(y)
v1=mean(x.^2)-mean(x)^2
v2=mean(y.^2)-mean(y)^2

cor=c/(sqrt(v1)*sqrt(v2))
```

g) Chaînes de Markov

L'instruction `grand(n,'markov',A',x0)` renvoit la liste des réalisations $(x_1, x_2, ..., x_n)$ des états des instants 1 à n : $X_1, X_2, ..., X_n$ d'une chaîne de Markov de matrice de transition A et partant de l'état initial $X_0 = x_0$.