1 Opérations et fonctions usuelles

1.1 Les symboles usuels

1.1.1 Le symbole d'affectation « = »

- `x=3` signifie que la valeur 3 est affectée à la variable `x` ($x \longleftarrow 3$).
- `x=x+3` signifie que le résultat de l'opération `x+3` écrase l'ancienne valeur de `x` et devient la nouvelle valeur de `x` ($x \longleftarrow x+3$).

Remarque : chronologiquement, la quantité du membre de droite est d'abord calculée, et est ensuite affectée à la variable indiquée à gauche.

1.1.2 Le délimiteur « ; »

En mode console, le « ; » marque la fin d'une instruction.

En mode éditeur sci, la fin d'une instruction peut être marquée par une nouvelle ligne.

1.1.3 Opérations algébriques

$+$ $-$ $*$ $/$

1.1.4 Nombres Usuels

`%e` (e) `%pi` (π) `%i` (i)

`ans` (variable qui stocke le dernier résultat calculé (« last answer »))

1.1.5 Comparaison de deux nombres

`<` `>` `<=` ($\leqslant$) `>=` ($\geqslant$) `<>` ($\neq$) `==` ($=$)

1.1.6 Connecteurs logiques

`&` (et) `|` (ou inclusif) `~` (non)

1.2 Fonctions usuelles

`log` (ln) `exp` `^` `cos` `sin` `tan`
`atan` (arctan) `sqrt` ($\sqrt{.}$) `floor` ($\lfloor . \rfloor$) `abs` ($|.|$)

1.3 Déclaration d'une fonction

- `deff("[y]=f(x)","y=...")` où `x` désigne un tableau de valeurs.
- `function y=f(x)  y=... endfunction`

2 Matrices

2.1 Matrices Lignes

- `A=[0 5 4]`
- `A=[1:100]` (`A=[1 2 ...100]`)
- `A=[1:0.5:100]` (`A=[1 1.5 2 ...100]`)
- `A(i)` : `i`-ème coefficient de la matrice `A`.
- `B=A(n1:n2)` donne la sous-matrice des coefficients d'indices `n1` à `n2`.

2.2 Matrices à plusieurs lignes

Une matrice à plusieurs lignes est obtenue par juxtaposition de matrices à une ligne, toutes de même longueur. Le séparateur est un « ; ».

Syntaxe `A=[L1;L2;...;Ln]`

- `A(i,j)` : coefficient `i`-ème ligne, `j`-ème colonne de la matrice `A`.

2.3 Taille d'une matrice

- `[n,p]=size(A)` matrice à `n` lignes et `p` colonnes.

2.4 Écriture et création de matrices

- `B=A(n1:n2,p1:p2)` donne la sous-matrice des lignes `n1` à `n2` et des colonnes `p1` à `p2`.
- `B=A(i,:)` ligne `i` de `A`.
- `B=A(:,j)` colonne `j` de `A`.
- `A=[B,C]` est la concaténation en colonne de `B` avec `C` si `B` et `C` sont de tailles compatibles (même nombre de lignes).
- `zeros(m1,m2)` : renvoie une matrice nulle à `m1` lignes et `m2` colonnes.
- `ones(m1,m2)` : renvoie une matrice remplie de « 1 » à `m1` lignes et `m2` colonnes.
- `eye(n,p)` : renvoie une matrice à `n` lignes et `p` colonnes avec des « 1 » sur la diagonale principale et des « 0 » partout ailleurs (matrice échelonnée).
- `eye(A)` : renvoie une matrice de même dimension que la matrice `A` avec des « 1 » sur la diagonale principale et des « 0 » partout ailleurs (matrice échelonnée).
- `A'` : donne la transposée de `A`.

2.5 Opérations sur les coefficients

- `A+1` : on ajoute `1` à tous les coefficients de la matrice `A`.
- `2*A` : tous les coefficients de `A` sont multipliés par `2`.
- `sqrt(A)`, `log(A)`,... : calcule les images par la fonction considérée de tous les coefficients de `A`.
- `.*` : on multiplie point par point les coefficients du premier objet avec les coefficients du deuxième objet (l'espace avant le point est important).
- `./` : on divise point par point les coefficients du premier objet avec les coefficients du deuxième objet. Par exemple : `1 ./A` calcule l'inverse de tous les coefficients de `A`, mais ne calcule pas l'inverse de `A` (cf. algèbre linéaire).
- `.^` : met à la puissance point par point les coefficients du premier objet par les coefficients du deuxième objet.
- `prod(A)` : produit de tous les coefficients de la matrice `A`.
- `sum(A)` : somme de tous les coefficients de la matrice `A`.

2.6 Algèbre linéaire

- `*` : produit matriciel classique.
- `inv(A)` ou `A^ -1` donne l'inverse de la matrice `A`.
- `rank(A)` donne le rang de la matrice `A`.
- `kernel(A)` donne une matrice dont les colonnes forment une base du noyau de `A`.
- `[X0,kerA]=linsolve(A,B)` résout le système linéaire `AX+B = 0` où `X0` est une solution particulière et `kerA` la solution du système homogène `AX = 0`.
- `spec(A)` : ensemble des valeurs propres de `A`. Par exemple `vp=spec(A)` retourne le tableau des valeurs propre et `[vp,X]=spec(A)` retourne valeurs et vecteurs propres de `A`.

2.7 Recherche d'éléments et tri

- `min(A)` : donne le plus petit des coefficients de la matrice réelle `A`.
- `max(A)` : donne le plus grand des coefficients de la matrice réelle `A`.
- `gsort(A)` : trie dans l'ordre décroissant les valeurs d'une matrice et `gsort(A,'g','i')` permet un tri dans l'ordre croissant.

  Avec des options `'r'` ou `'c'` à la place de `'g'`, on trie par lignes ou colonnes.

- `[ind,oc,info]=dsearch(A,tab)` : `A` est une matrice et `tab` une matrice ligne de valeurs rangées dans l'ordre croissant. `dsearch` recherche le nombre d'occurence dans les intervalles `]tab(i),tab(i+1)]` qui sont stockées dans le tableau `oc`.

  `[ind,oc,info]=dsearch(A,tab,"d")` recherche le nombre d'occurences de `tab(1)`, `tab(2)` ...
- `ind=find(A==s)` : recherche l'occurence `s` d'une valeur ou d'une chaine de caractère dans la matrice `A` et donne la liste des indices correspondants dans la matrice `ind`.
- `tabul(A)` : occurence d'apparition des valeurs d'une matrice classée dans l'ordre décroissant (écriture simplifiée de dsearch)

2.8 Fonctions statistiques

- `mean(A)` : fait la moyenne arithmétique de tous les coefficients de la matrice.
- `median(A)` : donne la médiane des coefficients de la matrice.
- `st_deviation(A)` : donne une estimation de l'écart-type d'une population dont les valeurs de la matrice forment un échantillon. Si la matrice représente la population totale, pour avoir l'écart-type il faut multiplier par un coefficient égal à $\sqrt{\frac{n-1}{n}}$.
- `variance(A)` : donne une estimation de la variance d'une population dont les valeurs de la matrice forment un échantillon. Si la matrice représente la population totale, pour avoir la variance il faut multiplier par un coefficient égal à $\frac{n-1}{n}$.
- `corr` : `[cov,m]=corr(x,y,1)` où `cov` est la covariance `cov(x,y)` et `m` un tableau de deux valeurs avec `m(1)` la moyenne des `x` et `m(2)` la moyenne des `y`. À partir de ces coefficients, on peut trouver les ajustements affines et le coefficient de corrélation entre les valeurs de `x` et `y`.
- `cumsum(A)` : somme cumulative des éléments d'un tableau. Par exemple `cumsum(A)` donne un tableau de mêmes dimension que `A` de sommes cumulées. `cumsum(A,1)` donne un tableau de sommes cumulées par colonnes et `cumsum(A,2)` donne un tableau de sommes cumulées par ligne.

3 Interface

3.1 Saisie au clavier et affichage à l'écran

3.1.1 Input

L'instruction `x=input("message")` permet de stocker dans la variable `x` la saisie réalisée par l'utilisateur. Le message doit être écrit entre guillemets.

3.1.2 Disp

L'instruction `disp(...)` permet d'afficher un message pouvant contenir éventuellement (même frequemment) le contenu d'une variable. L'affichage s'établit de droite vers la gauche. Par exemple `disp(x,"x=")`. Le séparateur entre les objets à afficher est la virgule « , ».

3.2 Représentations Graphiques

3.2.1 plot2d

`plot2d(x,y)` permet de tracer la courbe rejoignant les points de coordonnées `(x,y)` où `x` et `y` désignent des matrices.

Pour le tracé des courbes, il faut prendre l'habitude de travailler en matrice colonne : donc `plot2d(x',y')`.

Á cette instruction, il y a plusieurs options qui peuvent être associées :

- `plot2d(x,[y1,y2,...,yn])` pour tracer plusieurs courbes dans un même repère ;
- `plot2d(x,y,leg="...")` pour afficher une légende ;
- `plot2d(x,y,n)` où `n` désigne un nombre entier auquel correspond une couleur ;
- `plot2d(x,y,rect=[xmin,ymin,xmax,ymax])` délimite les valeurs minimales et maximales sur les axes.
- `plot2d(x,y,axesflag=n)` positionne le repère dans la fenêtre.

3.2.2 fplot2d

`fplot2d(x,f <opt>)` trace la courbe de la fonction `f` où `x` désigne ici un tableau de valeurs. les options sont les mêmes que `plot2d`.

3.2.3 plot3d et fplot3d

Même principe que `plot2d` et `fplot2d` mais en dimension 3. Permet donc le tracé des surfaces.

3.2.4 bar

L'instruction `bar(x,y)` réalise un diagramme en bâtons. Si l'on souhaite avoir plusieurs représentations sur le même repère on peut utiliser la fonction `bar(x,[y1,y2,...yn])` ; `x` doit être une matrice colonne.

3.2.5 histplot

L'instruction `histplot(x,y)` réalise l'histogramme des fréquences d'apparition de chaque valeur de `x`. La matrice `x` des valeurs doit être rangées dans l'ordre croissant et la matrice `y` est égale au nombre d'apparitions de chaque valeur de `x`.

3.2.6 pie

`pie(x)` réalise un diagramme en camembert où `x` est un tableau d'effectifs partiels ($\geqslant 1$).

4 Instructions conditionnelles et/ou répétitives

4.1 Instructions conditionnelles : if . . . then . . . <else . . .> end

La syntaxe conditionnelle `if...then ...<else ...> end` permet de réaliser une liste d'instructions à partir du moment où une condition est réalisée.

La syntaxe est, suivant la présence de `else` ou non :

```
if condition then
    instruction_1
    instruction_2
    ...
    instruction_n
end
```

```
if condition then
    instruction_A1
    instruction_A2
    ...
    instruction_An
else
    instruction_B1
    instruction_B2
    ...
    instruction_Bp
end
```

4.2 Boucles indexées : for. . . end

Lorsqu'il s'agit de répéter un certain nombre de fois une liste d'instructions, ce nombre de fois étant déterminé à l'avance, on utilise l'instruction `for...end` qui se présente ainsi :

```
for indice=rang_debut: pas : rang_fin
    instruction_1
    instruction_2
    ...
    instruction_n
end
```

Par défaut : `for indice=rang_debut:rang_fin` présente un pas d'incrémentation égal à 1.

4.3 Boucles conditionnelles : while. . . end

Lorsqu'il sagit de répéter un certain nombre de fois une liste d'instructions, dont le nombre de fois n'est pas déterminé à l'avance mais dépend d'une condition simple ou complexe alors on utilise l'instruction `while ...end` qui se présente ainsi :

```
while condition(s)
    instruction_1
    instruction_2
    ...
    instruction_n
end
```

5 Probabilités

5.1 Simulation du hasard

Tout repose sur la fonction `rand`.

- `rand` renvoie un nombre « au hasard » pris dans l'intervalle $[\,0\,;\,1\,[$.
- `rand(m,n)` renvoie une matrice à `m` lignes et `n` colonnes dont chaque coefficient est pris « au hasard » dans $[\,0\,;\,1\,[$.
- `rand("seed",getdate("s"))` : initialisation de la liste des nombres aléatoires

5.2 Simulation de variables aléatoires

5.2.1 La fonction grand...

La fonction `grand` permet de simuler le résultat d'une expérience aléatoire usuelle. La syntaxe est la suivante :

`Y=grand(n_ligne,n_colonne,"nom_loi",paramètre(s))`

qui retourne une matrice à `n_ligne` lignes et `m_colonne` colonnes dont chaque coefficient est le résultat d'une simulation.

5.2.2 ... pour les lois usuelles

- `grand(m,n,"uin",a,b)` : $\mathcal{U}(\llbracket a\,;\,b\rrbracket)$;
- `grand(m,n,"bin",N,p)` : $\mathcal{B}(\mathrm{N},p)$;
- `grand(m,n,"geom",p)` : $\mathcal{G}(p)$;
- `grand(m,n,"poi",lambda)` : $\mathcal{P}(\lambda)$;
- `grand(m,n,"unf",a,b)` : $\mathcal{U}([\,a\,;\,b\,[)$;
- `grand(m,n,"exp",1/lambda)` : $\mathcal{E}(\lambda)$;
- `grand(m,n,"gam",b,nu)` : $\Gamma(b,\nu)$;
- grand(m,n,"nor",m,s) : $\mathcal{N}(m,s^2)$

Remarque : `cdfnor` est la fonction de répartition de la loi normale (Φ).

- `grand(n,"markov",M,X0)` : permet la simulation d'une expérience aléatoire du type $\mathrm{A}_{n+1} = \mathrm{MA}_n$ (chaîne de Markov)