Probabilités et statistiques
Travaux pratiques avec Matlab

Stefan Le Coz [1]

1. Ce document est librement inspiré de polycopiés dont je disposais lors de son écriture (2005). À l'époque, je n'avais pas pris la peine de faire de bibliographie et je n'ai pas gardé trace des document dont je me suis inspiré. Malgré les apparences, il n'y a donc aucune prétention à l'originalité quant à la présentation qui est faite dans ce document, qui est plus une compilation des différentes sources dont je disposais à l'époque qu'un travail véritablement original.

Table des matières

Chapitre 1

Introduction à Matlab

1.1 Remarques préliminaires concernant l'apprentissage de Matlab

Un langage de programmation, tel Matlab, ne s'apprend pas tout à fait comme une théorie mathématique. En particulier, il ne faut pas espérer l'apprendre de manière très linéaire.

Ce chapitre se veut une aide pour commencer à apprivoiser Matlab. Il introduit, par petites touches, ce qui est utile pour programmer.

S'il ne faut retenir qu'une seule chose de cette introduction, c'est... la remarque 1.2 (À l'aide !).

Matlab est un logiciel commercial de calcul matriciel développé par la société MathWorks. Son nom est la contraction de "Matrix Laboratory". Il consiste essentiellement en un interpréteur de commandes, écrites dans un langage de programmation spécifique appelé langage Matlab. Les commandes Matlab sont saisies et interprétées ligne à ligne dans une fenêtre (console). Comme nous le verrons plus loin, elles peuvent également être regroupées dans un fichier dont le nom se termine par `.m`.

Les variables Matlab sont toutes des tableaux, "arrays" en anglais, définies au moment de leur affectation. Il n'y a donc pas besoin de les déclarer. Un nombre complexe (ou réel, ou entier) est un tableau de taille 1×1, un vecteur ligne est un tableau de taille $1 \times n$ et un vecteur colonne est un tableau de taille $n \times 1$. Le langage Matlab a été conçu pour faciliter les opérations sur les matrices.

Le langage Matlab permet de manipuler des données de différents types dont certains sont imbriqués : entiers, nombres réels, nombres complexes, caractères, booléens. Ces données peuvent être assemblées en tableaux de différents types imbriqués : vecteurs, matrices, tenseurs (matrices à plus de 2 dimensions). Les chaînes de caractères sont des vecteurs de caractères.

1.2 Matlab

En Matlab, les nombres réels sont représentés en virgule flottante avec 52 chiffres significatifs en base 2, soit un peu moins de 16 en base 10 (regarder `help eps`). Les opérations élémentaires sur les nombres réels sont :

$$+ \quad - \quad * \quad / \quad \backslash \quad \hat{}$$

qui représentent respectivement l'addition, la soustraction, la multiplication, la division à gauche et la division à droite, et enfin l'élévation à une puissance. Ces opérations s'appliquent également aux matrices, et dans ce cas, \ et / sont différentes.

Comme nous allons le voir, Matlab différentie minuscules et majuscules dans les noms de variables. L'affectation est notée `=`.

Matlab effectue ses calculs dans l'ensemble des tableaux à plusieurs dimensions, et donc en particulier les matrices, carrées ou non, et les vecteurs ligne et colonne. Bien entendu les opérations sur ces tableaux ne sont faites que lorsqu'elles ont un sens. En général, Matlab donne un sens assez intuitif aux opérations entre matrices. Cela dit, son langage contient de nombreuses spécificités que nous allons apprendre à utiliser.

Voici un exemple de code Matlab, à saisir ligne par ligne dans la fenêtre de commandes.

Exemple 1.1

```
a=0
```

On crée une variable réelle nommée `a`, initialisée à 0. Matlab affiche sa valeur une fois que l'on a appuyé sur la touche `Entr\'{e}e`. En suffixant par un point-virgule, on évite l'affichage de la valeur de `a`.

```
a=0;
```

Pour connaître le contenu d'une variable, il suffit d'invoquer son nom.

```
a
```

Matlab différencie majuscules et minuscules. Ainsi, on peut créer la variable `A`, différente de `a`.

```
A=1
```

Vous pouvez rappeler les commandes précédement exécutées au moyen des touches fléchées de votre clavier (haut et bas).
Voici un calcul compliqué à base des variables `a` et `A` précédentes.

```
A*a+cos(a)/(1+sqrt(1+A^2))
```

La fonction `sqrt` donne la racine carrée ("square root"). La variable spéciale `ans` contient la dernière réponse de Matlab qui n'a pas été affectée à une variable. On peut lister les variables actuellement définies avec la commande `whos`. On voit que pour Matlab, les variables `a` et `A` sont des matrices 1×1.

```
whos
```

Pour une liste plus succincte, on peut utiliser la commande `who`. On peut détruire une variable au moyen de la commande `clear`.

```
clear a % destruction de la variable a
```

Vérifions que la variable `a` n'existe plus.

```
who
```

On peut aussi détruire toutes les variables à l'aide de la commande `clear`. Pour obtenir de l'aide (en anglais !) sur une commande, on peut utiliser la commande `help`.

```
help who
```

La commande `helpwin` permet d'obtenir une liste des commandes Matlab classée par thème. Nous pouvons être amené à rencontrer certaine valeurs particulières. Tester les commandes suivantes.

```
1/0         % Infini positif Inf
-1/0        % Infini n\'{e}gatif -Inf
0/0         % Not a Number (NaN)
0*Inf       % Not a Number (NaN)
1/Inf       % Donne bien z\'{e}ro
pi          % Donne bien le nombre pi
help   pi   % Donne sa d\'{e}finition pour Matlab
i           % Racine carr\'{e}e (complexe) de -1.
help   i    % Sa d\'{e}finition pour Matlab.
```

Remarquons qu'il est quand même possible d'utiliser `i` comme variable...

Remarque 1.2 [À l'aide !] La commande `help` permet d'obtenir de l'aide sur les commandes Matlab, `help help` donne de l'aide sur l'aide ! Enfin, la commande `lookfor` permet de lister les commandes Matlab par mots clés.

Remarque 1.3 [Affichages] Si `X` est un tableau :

— la commande `disp(X)` affiche son contenu sans afficher son nom. S'il est vide, rien n'est affiché. La variable `X` vide est symbolisée par `[ ]`, on peut tester si `X` est vide grâce à la commande `isempty(X)`.

— la commande `display(X)` affichera le nom de la variable (`X` ici) ainsi que son contenu, même s'il est vide. Cette commande est utilisée automatiquement par Matlab lorsque qu'une expression ne se termine pas par un point-virgule.

Exercice 1.4 Tester et interpréter les commandes suivantes.

```
A= [2,4,8;3,9,27] A(2,3) b=A(2,:) c=A(:,3) B=A(1:2,1:2) A U=[1:20]
V=[1:2:20] V-U(1:2:20) W=[pi/2:1:pi]
```

Les commandes comme `sqrt` qui prennent un ou plusieurs arguments (ou paramètres) entre parenthèses constituent des fonctions. Nous verrons plus loin comment en créer de nouvelles.

1.3 Calculs matriciels élémentaires

Les tableaux suivants indiquent différents calculs possibles sur les matrices.

Commande	Sens
`A=[3,4;1,0]`	Matrice 2×2
`B=(A>0)`	B = matrice tq B_{ij}=1 si A_{ij}>0 et 0 sinon
`V=[5;5]`	Vecteur colonne de dimension 2, s'écrit aussi `[5,5]'`
`length (V)`	Renvoie la longueur du vecteur V
`size(A)`	La fonction `size` renvoie la taille de `A`

FIGURE 1.1 – Quelques exemples.

`A'`	transposée de A
`A*B`	Multiplication matricielle de A et B
`A+B`	Addition matricielle de A et B
`A^99`	Puissance matricielle
`expm(A)`	Exponentielle matricielle
`inv(A)`	Matrice inverse
`sqrtm(A)`	Racine carrée matricielle (cf. `help sqrtm`)
`logm(A)`	Logarithme matriciel (cf. `help logm`)
`A*V`	Image du vecteur colonne V par la matrice carrée A
`V'*A`	Devinez...
`A\V`	Solution du système linéaire AX=V (par pivot, et pas par inv(A))
`help slash`	Aide explicative sur la division matricielle précédente

FIGURE 1.2 – Opérations matricielles classiques.

`A=ones(n,n)`	Matrice de taille $n \times n$ dont tous les éléments valent 1
`B=zeros(n,n)`	Matrice nulle de taille $n \times n$
`C=eye(n,n)`	Matrice identité de taille $n \times n$. Que donne `eye(3,7)` ?
`[A,B,C]`	juxtaposition horizontale des matrices A, B et
`[A;B;C]`	juxtaposition verticale des matrices A, B et C
`N=0*J`	Astuce pour obtenir une matrice N nulle de même taille que J sans connaître la taille de J.
`sparse`	Création d'une matrice «creuse» (cf `help sparse`)

FIGURE 1.3 – Création de matrices.

`C=A.*B`	C(i,j)=A(i,j)*B(i,j)
`C=A./B`	C(i,j)=A(i,j)/B(i,j)
`C=A.3`	C(i,j)=A(i,j)*3
`C=3+A`	C(i,j)=3+A(i,j)
`C=3*A`	C(i,j)=3*A(i,j)
`C=A./3`	C(i,j)=A(i,j)/3
`C=cos(A)`	C(i,j)=cos(A(i,j))
`C=log(A)`	C(i,j)=log(A(i,j)). Ne pas confondre avec logm(A) !
`G=sqrt (A)`	C(i,j)=sqrt(A(i,j)). Ne pas confondre avec sqrtm(A) !
`C=exp(A)`	C(i,j)=exp(A(i,j)). Ne pas confondre avec expm(A) !
`C=abs(A)`	C(i,j)=|A(i,j)|

FIGURE 1.4 – Opérations entrée par entrée.

L'aide sur les opérations élémentaires s'obtient par la commande `help ops` et celle sur les opérations élémentaires matricielles par `help elmat`.

L'ajout d'un point devant un opérateur arithmétique indique à Matlbab que les opérations sur la matrice se font composantes par composantes. Vous pouvez faire `help arith` pour de l'aide sur les opérations arithmétiques et `help @` pour de l'aide sur les opérateurs en général et les caractères spéciaux.

Comme on vient de le voir, les sous matrices s'obtiennent en spécifiant des intervalles d'indices. Si i,j, k sont des entiers relatifs, alors :

— `i:j` est identique à $i, i+1, ..., j$,

— `i:k:j` est identique à $i, i+k, i+2k, ..., j$.

L'intervalle vide est représenté par la matrice vide `[ ]`. Une ligne ou une colonne entière peut être obtenue en utilisant le caractère `:` seul. Enfin, une matrice d'entiers `E` peut servir à spécifier les indices d'une matrice `A`, en écrivant `A(E)`. Il ne faut pas confondre les expressions de la forme `[...]`, qui permettent de fabriquer des matrices, avec celles de la forme `M(...)`, qui permettent de considérer une sous-matrice de la matrice `M`.

1.4 Calculs matriciels plus élaborés

Le tableau 1.4 présente des opérations matricielle plus élaborées que les précédentes. Remarquons que les opérateurs de type `sum` servent très souvent lorsqu'il s'agit d'éviter les boucles.

`det(M)`	Déterminant
`rank(M)`	Rang
`trace(M)`	Trace
`D=eig(M)`	Renvoie le vecteur colonne des valeurs propres de M
`[P,D]=eig(M)`	Diagonalise `M=P*D*P`$^{-1}$ avec D diagonale
`sum(V)`	Somme des éléments du vecteur V
`sum(M)`	Somme des colonnes de la matrice M (renvoie un vecteur ligne)
`sum(M,2)`	Somme les éléments de la matrice M selon la dimension 2
`sum(sum(M)`	Somme totale des éléments de la matrice M
`cumsum(V)`	Sommes cumulatives des entrées du vecteur V
`cumsum(M)`	Matrice des sommes cumulatives des colonnes de M
`cumprod(V)`	Produits cumulatifs des entrées du vecteur V
`cumprod(M)`	Matrice des produits cumulatifs des colonnes de M
`max(M)`	Renvoie un vect. ligne = au max sur chaque col . de M
`max(max(M))`	Maximum des entrée de la matrice M.

FIGURE 1.5 – Calculs matriciels élaborés

Pour l'aide sur les opérations matricielles élémentaires et sur les fonctions matricielles, on peut utiliser `helpwin elmat` ainsi que `helpwin matfun`

1.5 Graphiques

Le principe général des représentations graphiques est de se ramener à des calculs sur des matrices ou des vecteurs. Ainsi la repésentation d'une fonction de $\mathbb{R}$ dans $\mathbb{R}$ commencera par la création d'un vecteur d'abscisses, en général régulièrement espacées, auxquelles on applique la fonction pour créer le vecteur des ordonnées. La commande `hold on` permet de superposer des tracés successifs sur une même figure.

1.6 Entrées et sorties

La commande `input` permet de demander à l'utilisateur de saisir des valeurs de variables. La commande `pause` permet de stopper l'exécution de Matlab pendant un temps déterminé, cf. `help pause`. La forme spéciale `pause off` désactive les pauses tandis que `pause on` les réactive. La commande `save` permet de sauvegarder le contenu des variables en cours ainsi que leur nom dans un fichier, dont

`mean(Y)`	Moyenne arithmétique
`median(Y)`	Médiane
`std(Y)`	écart type normalisé en $N-1$
`plot(Y)`	Tracé du vecteur Y. Abscisses ? Ordonnées ?
`plot(Y,'r-')`	Autre tracé avec couleur et type de ligne
`plot (X,Y,'r-')`	Vous comprenez
`hist(Y)`	Histogramme (10 classes par défaut, cf. `help hist`)
`[Eff,Cl]=hist(Y,50)`	Histogramme à 50 classes de Matlab
`figure`	Création d'une nouvelle fenêtre graphique
`title ('Nom')`	Titre de la figure
`xlabel('Nom')`	Titre des abscisses
`ylabel('Nom')`	Titre des ordonnées
`imagesc(A);`	Trace la matrice, couleurs = valeurs relatives
`hold off`	On ne va pas utiliser la même fenêtre graphique
`contour(A)`	Trace les lignes de niveau

FIGURE 1.6 – Fonctions graphiques.

le nom est par défaut `matlab.mat`. Ce fichier peut être lu par la commande `load`, qui restaure donc toutes les variables.

1.7 Opérations logiques, boucles et exécutions conditionnelles

Pour Matlab, tout nombre peut être considéré comme une valeur logique, en identifiant les nombres non nuls à vrai (noté `1`) et le nombre zéro à faux (noté `0`). La commande `boolean` permet d'obtenir le booléen associé à un nombre vu comme une valeur logique. Les principales opérations sur les valeurs logiques sont :

— la négation logique `~` ,
— le "ou" logique `|` ,
— le "et" logique `&`.

On peut également utiliser respectivement les fonctions `or`, `not` et `and`. Il y en a d'autres, faites donc un `help ops`. On peut créer des valeurs logiques à partir de nombres avec les opérations de comparaisons, qui activent automatiquement la nature logique de leurs arguments :

— "a égal à b" s'écrit `a == b`
— "a différent de b" s'écrit `a ~= b`
— "a supérieur strictement à b" s'écrit `a > b`
— "a supérieur ou égal à b" s'écrit `a >= b`
— "a inférieur strictement à b" s'écrit `a < b`
— "a inférieur ou égal à b" s'écrit : `a <= b`

On peut également utiliser respectivement les fonctions `eq`, `ne`, `gt`, `ge`, `lt` et `leq`. Le code Matlab `R=randn(5,5);M=(R>=2.1)` crée une matrice `M` de même taille que `R` qui contient des 0 là où `R` est inférieure à 2.1 et des 1 là où `R` est supérieure ou égale à 2.1. C'est donc une matrice "booléenne". La commande `any(V)` renvoie 1 si au moins l'un des éléments du vecteur `V` est non nul, et 0 sinon. La fonction `all(V)` renvoie 1 si tous les éléments du vecteur `V` sont non nuls, et 0 sinon. Enfin la fonction `find(V)` renvoie les indices correspondants aux éléments non nul du vecteur `V`. Ainsi, `V(find(V>2))` renvoie les valeurs supérieures à 2 du vecteur `V`.

Les commandes Matlab `isreal`, `ischar`, `issparse`, `isnan`, `isinf`, `isfinite` permettent de tester la nature des variables. Ainsi, `isreal(a)` renverra 1 si la variable `a` est un réel et 0 sinon.

Le langage Matlab comprend les boucles du type `for` et `while` ainsi que les structures d'exécutions conditionnelles du type `if`. L'argument d'un `if` et d'un `while` est de type logique, ce qui n'est pas le cas de celui d'un `for`, qui est de type matriciel. La commande `break` permet de sortir de la boucle `while` ou `for` la plus interne.

1.8 Fonctions et fichiers .m

Les commandes Matlab peuvent toutes être considérées comme des fonctions, c'est-à-dire des entitées nommées, qui prennent des paramètres éventuels (arguments) et qui renvoient des résultats (valeurs de retour). Beaucoup de fonctions Matlab, comme `mean` par exemple, sont en réalité écrites en Matlab, et le code Matlab correspondant est stocké dans un fichier dont le nom se termine par `.m`. Pour `mean`, il s'agit de `mean.m`. Pour ajouter de nouvelles fonctions à Matlab, il nous suffit d'écrire de nouveaux fichiers de ce type. Vous aurez compris qu'un fichier `.m` ne contient qu'une seule fonction, qui a le même nom que le fichier, au suffixe `.m` près.

Voici un code qui définit une fonction `stat`, qui prend comme paramètre un vecteur `x` et qui renvoie sa moyenne et son écart type. Ce code devra être stocké dans le fichier nommé `stat.m` pour que Matlab fasse le lien avec la fonction `stat`. Le commentaire de la deuxième ligne constitue l'aide qui est affichée lorsque l'utilisateur tape `help stat`.

```
— function [moyenne,ecartype]=stat(x)
— %STAT Renvoie la moyenne et l'\'{e}cart type
  du vecteur pass\'{e} en argument."
— n=length(x);
— moyenne=sum(x)/n;
— ecartype=sqrt(sum((x-moyenne)/n);
— return % En fait, inutile en fin de fonction
```

Le fichier `stat.m` devra être placé dans un répertoire que Matlab scrutera. En général, Matlab cherche automatiquement dans le répertoire en cours. Pour afficher le répertoire en cours, utilisez la fonction `pwd`, pour lister son contenu,

utilisez la fonction `ls` et pour changer de répertoire courant, utilisez la fonction `cd`.

La commande Matlab `type` permet de lister le contenu du fichier `.m` d'une fonction. Ainsi, `type mean` va vous montrer le code source Matlab de la fonction `mean`. Un certain nombre de fonctions Matlab ne correspondent à aucun fichier `.m`, elles sont "internes" à Matlab pour une plus grande efficacité et l'on parle de fonctions "built-in". C'est par exemple le cas de la fonction `type` elle-même ! Cette séparation entre fonctions internes et externes se retrouve dans la plupart des interpréteurs de langages. La commande `exist` permet de connaître le type d'une commande ou d'une variable, cf. `help exist`.

Sur la plupart des systèmes informatiques, les fichiers sont organisés en une structure arborescente de répertoires qui contiennent des fichiers et des sous-répertoires. La commande Matlab `which` donne le chemin du fichier `.m` associé à une fonction Matlab, lorsqu'il existe. Par exemple, `which mean -ALL` affichera tous les fichiers `mean.m` avec leurs chemins. Seul le premier sera utilisé par Matlab ! La commande `what` liste les fichiers `.m` du répertoire en cours. Faites `help what` et `help which` pour plus d'information.

Matlab a besoin de savoir où chercher les fichiers `.m` dans l'arborescence des fichiers du système. Il recherchera automatiquement dans une liste de réperoires appelée "PATH". La commande `path` vous liste les répertoires du "PATH". Les commandes `addpath` et `rmpath` permettent d'ajouter et d'enlever des répertoires de la liste "PATH". Ces modifications ne sont pas permanentes et sont perdues lorsque vous quittez Matlab.

Par défaut, la liste "PATH" contient les répertoires de Matlab, qui contiennent eux-même les fichiers `.m` qui constituent Matlab. Afin d'accélérer la lecture des fichiers `.m`, Matlab maintient en permanence une base de données des fichiers `.m` avec le répertoire associé qui les contient. La commande `rehash` permet de mettre à jour cette base de données en relisant la liste "PATH".

Il faut enfin savoir que Matlab cherche d'abord les fichiers `.m` des fonctions dans le répertoire courant ("working directory"). Répétons-le, la commande `pwd` affiche le répertoire courant ("print working directory"), la commande `cd` change de répertoire courant ("change directory"), et les commande `ls` ou `dir` listent le contenu du répertoire courant.

En général, vous n'aurez pas à modifier la liste "PATH". Sa modificaton permanente dépend du système utilisé (Unix, MS-Windows,...).

1.9 Récursivité

Le langage Matlab autorise la récursivité, comme le montre l'exemple archi-classique suivant :

```
function fn=fact(n);
% fact cacule de fa\c{c}on r\'{e}cursive (et particuli\`{e}rement inefficace)
```

```
% la factorielle de n.
if (n<=0), fn=1; else fn=n*fact(n-1); end
```

Exercice 1.5 En utilisant la fonction `prod`, écrire une fontion calculant $n!$ de façon non récursive (et beaucoup plus efficace).

1.10 Fonctions en ligne

Pour des fonctions très légères, on peut procéder autrement que par la création d'un fichier `.m` : la directive `inline` permet de créer des fonction nouvelles à la volée. Voici un exemple :
On définit une nouvelle fonction `trisin` qui prend 3 paramètres a, b et c
`trisin=inline('sin(a)*sin(b)*sin(c)');`
On teste notre nouvelle fonction :
`trisin(pi/2,pi/4,pi/2)`

Matlab détermine automatiquement le nombre de paramètres de la fonction. Bien entendu, une fois que vous quitterez Matlab, cette fonction sera perdue car elle n'est pas sauvegardée dans un fichier.

Exercice 1.6 Apprenez à utiliser la fonction `fplot` pour tracer les graphes de fonctions définies avec `inline`.

Chapitre 2

Statistique descriptive

Exercice 2.1 On se donne les séries statistiques suivantes. Écrire un programme donnant pour chacune d'entre elles une représentation graphique des données (diagramme en baton ou histogramme selon les cas), tracant la fonction de répartition, et indiquant les caractéristiques de positions (mode, médiane, moyenne, quartiles) et de dispertion (étendue, variance, écart-type).

Ménages	Effectifs	Fréquences
1 personne	5 845 140	0,2713
2 personnes	6 366 948	0,2957
3 personnes	3 821 700	0,1774
4 personnes	3 371 484	0,1565
5 personnes	1 439 144	0,0668
6 personnes ou plus	695 736	0,0323
Somme	21 542 152	1,0000

FIGURE 2.1 – Ménages suivant le nombre de personnes du ménage en 1990, en France.

Exercice 2.2 [Séries statistiques à deux variables] On se donne la série statistique double suivante.

Écrire un programme donnant une représentation graphique des données (nuage de points), traçant la droite de régression (en utilisant la méthode des moindres carrés) et indiquant le coefficient de corrélation entre les deux variables.

Classes d'âge	Effectifs	Classes d'âge	Effectifs	Classes d'âge	Effectifs
[15,17[	800	[29,31[	51200	[41,43[	800
[17,19[	7 000	[31,33[	32 000	[43,45[	200
[19,21[	14 350	[33,35[	23 200	[45,47[	80
[21,23[	28 000	[35,37[	16 000	[47,49[	40
[23,25[	44 000	[37,391	8 000		
[25,271	54 800	[39,41[	2 400		
[27,29[	56000				

FIGURE 2.2 – Naissances en France en 1994 en fonction de l'âge de la mère (en millier).

Firmes	CA	Firmes	CA	Firmes	CA
General Motors	860	Nestlé	231	PSA	166
Ford Motor	713	FIAT	226	BASF	159
Exxon	563	Sony	216	ABB	158
Royal Dutch Shell	526	Honda Motor	215	Pepsico	158
Toyota Motor	441	Elf Aquitaine	208	PEMEX	156
Hitachi	412	NEC	205	Nippon Steel	156
Matsushita	377	Daewoo	197	Mitsubishi Heavy I.	155
General Electric	359	Du Pont de Nemours	194	Amoco	150
Daimler Benz	356	RWE	191	Bayer	148
IBM	355	Japan Tobacco	190	Nippon Oil	144
Mobil	331	Philips	186	BMW	144
Nissan Motor	317	Mitsubishi Motors	185	Hewlett-Packard	139
Philip Morris	298	Texaco	185	Kansai Electric P.	138
Chrysler	290	EDF	183	Total	137
Siemens	289	Renault	179	Conagra	131
British Petroleum	281	Fujitsu	177	Motorola	123
IRÏ	279	Mitsubishi Electric	176	Petroleos de Venezuela	123
Volskwagen	273	ENI	174	Bat industries	122
Tokyo Electric P.	271	Chevron	172	Boeing	122
Toshiba	260	Hoeschst	170	Mazda Motor	120
Unilever	251	Procter & Gamble	168	Thyssen	119
VEBA	243	Alcatel Alsthom	168		

FIGURE 2.3 – Chiffres d'affaires (en milliards de francs), en 1994, des principaux groupes industriels mondiaux.

X_i	29	4	69	65	98	55	40	19	62	73	37	0	41	75	79
Y_i	26	4	65	66	90	56	43	26	66	76	35	2	45	68	86

Chapitre 3

Simulation de variables aléatoires

Considérons une suite de chiffres x_1, x_2, ..., x_n de l'intervalle $[0, 1]$ ou d'entiers entre 0 et 9. Quand peut-on dire qu'elle est constituée de réalisations indépendantes d'une loi uniforme sur $[0, 1]$ (resp. sur $\{0, ..., 9\}$) ? Et bien dans tout les cas bien sûr. Cependant, l'intuition nous indique que ce sont les propriétés "statistiques" des suites qui sont importantes, et pas celles d'une suite donnée... Pour certains, une suite qui peut être générée par un algorithme n'est pas aléatoire. De ce point de vue, une suite finie n'est jamais aléatoire !

Pour nous, simuler une loi de probabilité $\mathcal{L}$ consistera à écrire un programme informatique pouvant générer des suites finies dont on considérera que ce sont des réalisations indépendantes de loi $\mathcal{L}$. Bien entendu, comme nous ferons appel le plus souvent à des algorithmes déterministes, les suites générées ne seront pas du tout aléatoires et encore moins indépendantes. On parlera de suites pseudo-aléatoires. Cependant, elles se montrent suffisantes dans la pratique et l'on pourra vérifier expérimentalement qu'elles ont de "bonnes" propriétés en utilisant les tests statistiques.

La loi uniforme est la première loi que l'on simule. Diverses méthodes permettent ensuite de simuler à partir de la loi uniforme, un grand nombre de lois, plus ou moins quelconques. Dans la pratique, on met en œuvre des algorithmes rapides de génération de suites pseudo-aléatoires.

Matlab permet de simuler une loi uniforme (resp. normale) via la fonction `rand` (resp. `randn`).

`rand`	renvoie un nombre selon la loi uniforme entre 0 et 1
`randn`	renvoie un nombre selon la loi normale
`rand(4,4)`	Une matrice 4x4 dont les entrées sont des réalisations de loi uniforme sur [0,1]
`randn(4,4)`	Idem mais avec la loi normale
`rand(1000,1)`	1000 réalisations pseudo-uniformes et pseudo-indépendantes

FIGURE 3.1 – Uilisation de **rand et randn**

3.1 Loi uniforme et simulation

Les générateurs de nombres aléatoires sont très importants dans les applications. Ils permettent d'une part de simuler des phénomènes dont la modélisation fait appel à des lois de probabilités et d'autre part de sécuriser les échanges de données, dans les réseaux téléinformatiques par exemple. Dans la pratique, les générateurs aléatoires sont basés sur des algorithmes déterministes ou encore sur une source d'"entropie" extérieure difficilement prédictible liée au fonctionnement de la machinerie informatique (par exemple le temps séparant deux interruptions du processeur), ces deux approches pouvant être combinées dans un même processus.

Simuler la loi uniforme consiste à produire une suite de nombres qui peuvent être considérés comme autant de réalisations indépendantes de variables aléatoires uniformes sur $[0,1]$. Bien entendu, un algorithme est toujours déterministe lorsqu'on le connaît, et il vaut donc mieux parler de générateur pseudo-aléatoire de la loi uniforme. Mathématiquement, les n sorties successives d'un tel générateur seront considérées comme la donnée de $U_1(\omega), ..., U_n(\omega)$ pour un $\omega \in \Omega$ où les U_i sont des v.a.r. i.i.d. $U_i : (\Omega, \mathcal{F}, P) \to [0,1]$ de loi uniforme.

La mémoire des ordinateurs étant finie, il est impossible de stocker des suites infinies. En conséquence, les nombres réels sont toujours approximés, avec une certaine précision. Soit donc $p \in \mathbb{N}^\star$ qui va jouer le rôle de précision. Un réel x_n dans [0,1] s'écrira y_n/p où y_n est un entier dans $\{0, ..., p\}$. Un algorithme très usité pour générer les y_n consiste à utiliser la relation de récurrence $x_n - y_n/p$ où $y_{n+1} \equiv a y_n[m]$, a et m sont des entiers et m est assez grand.

En général, la période de $(y_n)_n$ est plus petite que m. Plus précisément, si a et m sont premiers entre eux, et si $0 < y_0 < m$, alors y_n n'est jamais nul et la période de $(y_n)_n$ n'est rien d'autre que l'ordre de a dans $\mathbb{Z}_m$. On rappelle que d'après le (petit) théorème de Fermat-Euler, l'ordre de a divise l'indicateur d'Euler de a (qui vaut $m-1$ quand m est premier).

Matlab 4 utilise un algorithme de ce type pour implémenter la fonction `rand` mais les versions plus récentes de Matlab utilisent un algorithme différent et plus performant. Dans certaines bibliothèques de calcul scientifique, on utilise $(a, m) = (13^{13}, 2^{59})$ ou encore $(a, m) = (7^5, 2^{31} - 1)$.

3.2 Simulation de lois discrètes

Soit $p_1, ..., p_n$ des nombres dans $[0,1]$ tels que $p_1 + \cdots + p_n = 1$. En partitionnant l'intervalle $[0,1]$ en morceaux adjacents de longueurs $p_1, ..., p_n$, il vient :

Théorème 3.1 (Simulation d'une loi discrète à support fini) *Soit U une v.a.r. de loi uniforme sur $[0,1]$ et $P := p_1\delta_{x_1} + \cdots + p_n\delta_{x_n}$ une loi discrète où les x_i sont tous différents. Alors la v.a.r.*

$$x_1 I_{\{U<p_1\}} + x_2 I_{\{p_1 \leqslant U \leqslant p_1+p_2\}} + \cdots + x_n I_{\{p_1+\cdots p_{n-1} \leqslant U \leqslant 1\}}$$

suit la loi P.

Exercice 3.2

(1) Écrire une fonction implémentant le résultat du Théorème 3.1.

(2) Simuler, sans utiliser d'instruction conditionnelle, la loi uniforme discrète sur $\{1, ..., 10\}$.

(3) Idem pour la loi binomiale $\mathcal{B}(n, p)$.

(4) Montrer que la méthode issue du Théorème 3.1 s'étend sans difficulté aux lois discrètes à support dénombrable. Donnez un programme d'illustration.

(5) Simuler la loi de Poisson.

(6) Illustrer graphiquement la convergence de la loi binomiale $\mathcal{B}(n,\, p)$ vers la loi de Poisson $\mathcal{P}(\lambda)$ lorsque $np \to \lambda$.

(7) Illustrer graphiquement la convergence de la loi binomiale $\mathcal{B}(n,\, p)$ vers la loi normale $\mathcal{N}(np,\, p(1-p))$ (théorème de Moïvre-Laplace).

3.3 Simulation de lois par la méthode du rejet

On attribue parfois cette méthode à J. von Neumann. Soit une loi $\mathcal{L}$ de densité f sur $\mathbb{R}$, que l'on désire simuler. Supposons que f est continue et à support compact $[a, b]$. Le graphe de f est donc contenu dans un rectangle $[a, b] \times [0, M]$. On considère alors des vecteurs aléatoires indépendants $((X_n, Y_n))_n$ de loi uniforme sur $[a, b] \times [0, M]$. On définit ensuite T comme le plus petit n tel que $f(X_n) \leqslant Y_n$. Alors on montre que T est fini presque sûrement et que X_T a pour loi $\mathcal{L}$.

Exercice 3.3 Simuler la loi uniforme sur le disque unité dans $\mathbb{R}^2$ par la méthode du rejet.

3.4 Simulation des lois gaussiennes

3.4.1 Méthode de Box-Müller

On utilise le résultat suivant

Proposition 3.4 *Soient U_1 et U_2 deux v.a.r. indépendantes de loi uniforme $\mathcal{U}([0,\, 1])$. Posons*

$$\begin{cases} X_1 = \sqrt{-2\ln(U_1)}\cos(2\pi U_2), \\ X_2 = \sqrt{-2\ln(U_1)}\sin(2\pi U_2). \end{cases}$$

Alors X_1 et X_2 sont deux v.a.r. indépendantes de loi normale centrée réduite $\mathcal{N}(0,\, 1)$.

Exercice 3.5

(1) En utilisant la Proposition 3.4, écrire une fonction de paramètre n simulant n réalisations de la loi normale centrée réduite $\mathcal{N}(0,\, 1)$.

(2) Écrire une fonction de paramètres n, m et σ simulant n réalisations d'une loi normale $\mathcal{N}(m, \sigma)$.

(3) Sachant qu'une v.a. suivant une loi du χ^2 à ν degrés de liberté peut être vu comme la somme de ν v.a. iid de loi normale $\mathcal{N}(0,\, 1)$, simuler n réalisations d'une v.a. de loi $\chi^2(\nu)$. Faire varier ν. Qu'obtient-on pour ν grand ($\nu > 30$).

3.4.2 Méthode polaire

Proposition 3.6 *Soient (U_1, U_2) un couple de loi uniforme sur le disque unité. Posons*

$$\begin{cases} X_1 = U_1 \sqrt{\dfrac{-2\ln(U_1^2 + U_2^2)}{U_1^2 + U_2^2}}, \\ X_2 = U_2 \sqrt{\dfrac{-2\ln(U_1^2 + U_2^2)}{U_1^2 + U_2^2}}. \end{cases}$$

Alors X_1 et X_2 sont deux v.a.r. indépendantes de loi normale centrée réduite $\mathcal{N}(0,\, 1)$.

Exercice 3.7 En utilisant la Proposition 3.6, écrire une fonction de paramètre n simulant n réalisations de la loi normale centrée réduite $\mathcal{N}(0,\, 1)$ (Indication : simuler d'abord la loi uniforme sur le disque unité par la méthode du rejet).

3.4.3 Quelques précisions

Ces méthodes sont à la fois très classiques et... très lentes, si bien que dans la pratique, on utilise les méthodes du type "Marsaglia", qui semblent être de loin les plus performantes. Il est préférable d'utiliser la fonction Matlab `randn` qui est déjà toute programmée et qui fait appel à une méthode plus efficace. En effet, la méthode de simulation de Box-Müller est coûteuse en temps de calcul car elle nécessite l'évaluation des fonctions log, $\sqrt{\ }$ et cos. On peut se passer du cos en faisant appel à la méthode polaire, qui est celle utilisée dans Matlab 4 pour implémenter la fonction `randn`. Elle demeure cependant assez lente, en raison à la fois des 21% de rejet et de l'évaluation des fonctions log et $\sqrt{\ }$.

Les versions 5 et 6 de Matlab utilisent plutôt une méthode due à Marsaglia, appelée "algorithme du Ziggurat". Elle consiste à utiliser la méthode du rejet conjointement à des tables donnant une approximation de la densité gaussienne par des rectangles. Ces tables étant précalculées, la génération de nombres aléatoires gaussiens est très rapide, quasiment aussi rapide que celle des nombres aléatoires uniformes.

3.5 Simulation de lois par leur fonction de répartition

Soit $\mathcal{L}$ une loi sur $\mathbb{R}$ de fonction de répartition F. Si G désigne l'inverse continue à gauche de F, définie pour tout y dans $[0,1]$ par

$$G(p) := \inf\{x \,:\, F(x) \geqslant p\},$$

alors, pour toute v.a. uniforme U sur $[0,1]$, la v.a. $G(U)$ est de loi $\mathcal{L}$. Par exemple, pour la loi de Cauchy, de densité $(\pi(1+x^2))^{-1}$ sur $\mathbb{R}$, on a $F(x) = \arctan(x)/\pi + 1/2$ et $G(p) = \tan(\pi(p-1/2))$.

Exercice 3.8 [Simulation de la loi exponentielle] En utilisant le résultat suivant, simuler 100 réalisations de la loi exponentielle.

Proposition 3.9 *Soit U une v.a.r. de loi uniforme sur $[0,1]$ et $\lambda > 0$ un réel. Alors la v.a.r. $-\lambda^{-1}\log(U)$ suit la loi exponentielle de paramètre λ.*

Cette méthode s'applique bien entendu aux lois de Weibull, dont les lois exponentielles ne constituent qu'un cas très particulier.

Solution 1

```
lambda=0.5;clf;hold on; title('Simulation d'une loi
exponentielle') ylabel('Densit\'{e}');xlabel('Valeurs');
[E,C]=histo(-log(rand(100,1))/lambda,100,0,1)
plot(C,lambda*exp(-lambda*C),'k-*')
legend('Empirique','Th\'{e}orique')
```

3.6 Étude de la planche de Galton

La planche de Galton est une planche verticale sur laquelle sont plantées n lignes de clous disposés en pyramide : 1 clou sur la première ligne, 2 clous sur la deuxième ligne,..., n clous sur la n-ième ligne. Des billes sont lâchées l'une après l'autre à partir du sommet. A chaque clou rencontré, la bille a

— une probabilité $p = 1/2$ de passer à droite (succès),

— une probabilité $1 - p = 1/2$ de passer à gauche (échec).

Sous la planche, les billes tombent dans $n+1$ compartiments, numérotés de 0 à n de gauche à droite, dans lesquels elles s'empilent.

Soit X la variable aléatoire qui à chaque bille associe le numéro du compartiment atteint. Ainsi, l'événement $(X = k)$ est réalisé quand la bille passe k fois à droite des clous (k succès). Il est évident que la v.a. X suit la loi binomiale $\mathcal{B}(n,p)$.

Exercice 3.10

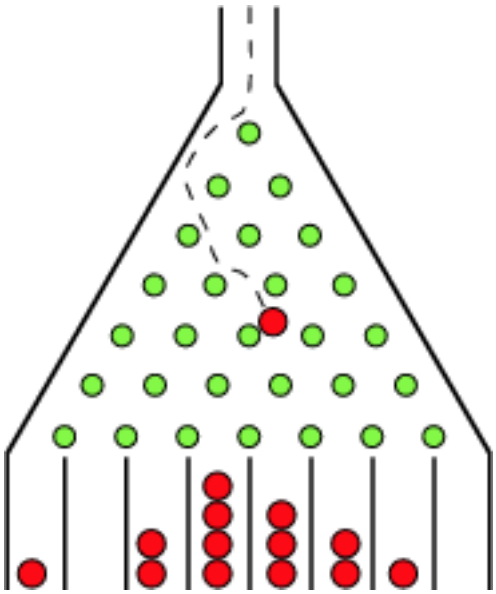

FIGURE 3.2 – Planche de Galton à 7 lignes

1. Écrire un programme de simulation de la planche de Galton. Ce programme consiste à
   (a) visualiser l'évolution des effectifs expérimentaux de billes tombées dans chaque compartiment (lors de la chute des billes l'une après l'autre),
   (b) donner les effectifs théoriques des billes tombées dans chaque compartiment,
   (c) comparer graphiquement les piles de billes tombées dans les compartiments avec les piles de billes calculées théoriquement.
2. Expliquer théoriquement la symétrie de la répartition des effectifs dans les compartiments.
3. Changer la valeur de p et expliquer les résultats.

Pour visualiser un joli programme d'illustration de la planche de Galton, vous pouvez consulter le site suivant.

http ://www-sop.inria.fr/mefisto/java/tutorial1/node11.html

Remarque 3.11 Il existe d'autres méthodes pour simuler de façon "concrète" la loi binomiale, notamment la machine de Bittering. On peut également réaliser d'autres expériences pratiques où les probabilités permettent de mesurer certaines grandeurs (aiguille de Buffon, cercle répétiteur,...)

Chapitre 4

Statistique inférentielle

4.1 Estimations et intervalles de confiance

Soit X une variable aléatoire sur une population *dont on ne peut faire une étude complète*. On suppose que la variable aléatoire X est de moyenne et de variance finie, et on note

$$E[X] = \mu\,; \qquad V[X] = \sigma^2.$$

Notons que ces paramètres sont inconnues a priori. On se propose d'une part, de donner une estimation ponctuelle de ces paramètres ; d'autre part, de donner un intervalle de confiance de μ au niveau de confiance $1-\alpha$. Pour cela, nous nous appuyerons sur les deux résultats théoriques suivants.

Théorème 4.1 (Loi faible des grands nombres) *Soit $(X_n)_{n\in\mathbb{N}}$ une suite de variables aléatoires indépendantes de même loi et d'espérance μ finie. Alors*

$$\frac{X_1 + \cdots + X_n}{n} \longrightarrow \mu \text{ en probabilité lorsque } n \to +\infty.$$

Remarque 4.2 On dit qu'une suite $(S_n)_n$ de v.a. converge en probabilité vers μ lorsque

$$\forall \varepsilon > 0 \quad \lim_{n\to+\infty} P(|S_n - \mu| > \varepsilon) = 0.$$

Théorème 4.3 (Théorème central-limite) *Soit $(X_n)_{n\in\mathbb{N}}$ une suite de variables aléatoires indépendantes de même loi, d'espérance μ et de variance σ^2 finies. Posons*

$$\forall n \in \mathbb{N}^\star \qquad \overline{X}_n := \frac{X_1 + \cdots + X_n}{n} \qquad \text{et} \qquad Z_n := \frac{\sqrt{n}}{\sigma}(\overline{X}_n - \mu).$$

Alors la suite $(Z_n)_n$ converge en loi vers la loi normale $\mathcal{N}(0,1)$.

Remarque 4.4 On dit qu'une suite $(Z_n)_n$ de v.a. converge en loi vers la loi normale lorsque

$$\forall a,\, b \in \mathbb{R} \text{ tel que } a < b, \text{ on a } \lim_{n\to+\infty} P(a < Z_n < b) = \frac{1}{\sqrt{2\pi}} \int_a^b e^{-x^2/2} dx.$$

4.1.1 Estimations ponctuelle

Le problème qui se pose est de donner, à partir d'un n-échantillon $(x_1, ..., x_n)$, une estimation des valeurs caractéristiques d'une v.a. définie sur une population. Le tableau suivant donne une première approche du problème.

	valeur théorique	valeur empirique
Moyenne	μ	$m_e = \frac{1}{n}\sum_{i=1}^{n} x_i$
Variance	σ^2	$\sigma_e^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - m_e)^2$

FIGURE 4.1 – Deux estimateurs

Mais ces estimateurs ne sont pas tous les "meilleurs" (en certain sens que nous ne chercherons pas à préciser). Le tableau suivant donne les "meilleurs" estimateurs des paramètres précédents.

	valeur théorique	"meilleur" estimateur
Moyenne	μ	m_e
Variance	σ^2	$\sigma_e^2\sqrt{\frac{n}{n-1}}$

FIGURE 4.2 – "Meilleurs" estimateurs

4.1.2 Intervalles de confiance

À tout n-échantillon $(x_1, ..., x_n)$ de moyenne empirique m_e on associe l'intervalle de confiance au niveau de confiance $1 - \alpha$ de la moyenne μ suivant :

$$I_n^\alpha := \left[m_e - t_\alpha \frac{\sigma}{\sqrt{n}},\ m_e + t_\alpha \frac{\sigma}{\sqrt{n}} \right],$$

où t_α est le réel défini par $\Pi(t_\alpha) = 1 - \frac{\alpha}{2}$, avec Π la fonction de répartition de la loi normale $\mathcal{N}(0,\, 1)$. Noter que si la variance σ^2 est inconnue on la remplace par son "meilleur" estimateur $\sigma_e^2\sqrt{\frac{n}{n-1}}$. Remarquons que l'inégalité de Bienaymé-Tchebychev permet également de trouver des intervalles de confiance, mais de façon bien moins précise que le théorème central-limite.

Remarque 4.5 Le réel t_α s'obtient par le code Matlab `sqrt(2)*erf(inv(1-a))`.

4.1.3 Simulations

Les objectifs de cette simulation sont

1. montrer expérimentalement le caractère relatif des intervalles de confiances ;
2. comprendre la notion de confiance qu'on peut leur attacher.

Exercice 4.6 [Comparaison du pourcentage d'intervalles de confiance contenant effectivement μ avec le seuil de confiance.]

1. Simuler m échantillons de tailles n d'une variable aléatoire suivant une loi de votre choix.
2. Pour un niveau de confiance $1-\alpha$ fixé, déterminer m intervalles de confiance de μ à partir des m échantillons.
3. Calculer le pourcentage d'intervalles contenant réellement μ.
4. Donner une illustration graphique de ces résultats.

4.2 Test du χ^2

4.2.1 Ajustement par la loi binomiale

Exercice 4.7 Un prélèvement de 100 échantillons de même effectif 40 pièces a montré l'existence d'un certain nombre de pièces défectueuses. Les résultats obtenus sont :

Nb de pièces défectueuses	Effectifs observés
0	28
1	40
2	21
3	7
4 ou plus	4

1. Tracer l'histogramme de la répartition de l'échantillon.
2. Quel type de loi cela inspire-t-il pour ajuster le phénomène ?
3. Déterminer les paramètres de la loi supposée et tester l'ajustement.

4.2.2 Ajustement par la loi de Poisson

Exercice 4.8 On observe les nombres de vols d'objets dans un immeuble de 4 étages. Les données sont les suivantes.

1. Est-il raisonnable de supposer que les vols suivent une loi de Poisson ?

Étage de l'immeuble	Nombre d'objets volés
0	545
1	325
2	110
3	15
4	5

2. Peut-on supposer que le nombre d'objets volés suit une loi de Poisson de paramètre $\lambda = 1$?
3. Si la réponse est oui pour la question 1., donner une estimation expérimentale du paramètre λ ?

4.2.3 Ajustement par la loi normale

Exercice 4.9 A l'école X, il est connu que grâce aux différentes notes (dont on fait une moyenne pondérée) on détermine pour chaque étudiant sa note finale. Sur l'ensemble de la promotion, la moyenne est de 10 sur 20 (précisément) ; mais cela cache bien des disparités :

Note finale	Nombre d'étudiants
Moins de 5	2
De 5 à moins de 7.5	21
De 7.5 à moins de 9.75	36
De 9.75 à moins de 10.25	30
De 10.25 à moins de 12.5	37
De 12.5 à moins de 15	23
15 et plus	1

1. Tracer l'histogramme de répartition des étudiants. Quel type de loi cela inspire-t-il pour ajuster le phénomène ?
2. **Droite de Henry** : Le principe de la droite de Henry repose sur l'hypothèse, supposée vérifiée, de normalité d'un phénomène. S'il est acquis que la v.a. X suit une loi normale $\mathcal{N}(m, \sigma)$, la v.a. $T = (X - m)/\sigma$ suit la loi $\mathcal{N}(0, 1)$. Ceci s'apparente à une équation de droite de pente $1/\sigma$ et d'ordonnée à l'origine $-m/\sigma$. Lorsqu'on n'est pas sûr de la normalité du phénomène, il faut représenter graphiquement les points (x_i, t_i) et constater s'ils sont ou non alignés. Les valeurs de t_i sont obtenues à partir des valeurs de $\Pi(t_i)$ assimilées aux valeurs de fréquences cumulées croissantes.
   (a) Compléter le tableau suivant. Pour déterminer les t_i à partir de $p_i = \Pi(t_i)$ on utilise la fonction inverse qu'on a déjà vu $t_i = \sqrt{2}\mathrm{erfinf}(2\mathrm{p_i} - 1)$

Note	Nb étudiants	Effectifs cumulés	Fréquences cumulées	t_i
Moins de 5	2			
De 5 à 7.5	21			
De 7.5 à 9.75	36			
De 9.75 à 10.25	30			
De 10.25 à 12.5	37			
De 12.5 à 15	23			
15 et plus	1			

(b) Tracer le nuage de points (x_i, t_i) : $(5, t_1), (6.25, t_2), ..., (15, t_6)$ (on prend les milieux des classes sauf pour les extrémités).

(c) Déterminer la droite de Henry qui n'est autre que la droite de régression de la série (x_i, t_j) (Pour cela, on peut utiliser la commande Matlab `polyfit` ; voir `help`). En déduire les caractéristiques de la loi normale supposée : $m = E(X)$ et $\sigma = \sigma(X)$ (graphiquement ou (et) par calcul).

3. Tester à l'aide du χ^2 la validité de cette hypothèse (loi normale) en utilisant la moyenne et l'écart type obtenus ci-dessus. N'oublier pas de déterminer le degré de liberté de la loi χ^2 et de regrouper les classes de faible effectif (moins de 5).