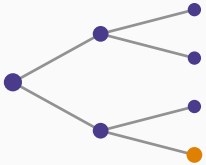
Probabilités

Chapitre 22 : Mise en œuvre sur R et Python



Jaouad Madkour

Faculté d'Économie et de Gestion de Tanger · 27 août 2026

Plan du chapitre

Le programme

Ce que ce chapitre reprend

Les vingt et un chapitres, en quelques lignes

Ce qu'on refait	Chapitres
Dénombrement	3 à 5
Lois discrètes et leurs quantiles	13 à 15
Lois continues et la table normale	16 à 19
Simulation et vérification des théorèmes	20, 21

Le but n'est pas d'apprendre un logiciel

C'est de **vérifier** les résultats calculés à la main et de comprendre la logique de nommage des fonctions — qui est la même partout.

Savoir lire une table normale reste utile : c'est ce qui permet de repérer un résultat aberrant, qu'aucun logiciel ne signalera.

La logique de nommage

Quatre lettres, et tout est dit

La convention de R

Préfixe	Rôle	Question
`d`	densité ou $P(X = k)$	quelle probabilité pour cette valeur ?
`p`	répartition $F(x)$	quelle probabilité en dessous ?
`q`	quantile $F^{-1}(\alpha)$	quelle valeur pour cette probabilité ?
`r`	tirage aléatoire	simuler

On accole le nom de la loi

`binom`, `pois`, `geom`, `unif`, `exp`, `norm`, `chisq`, `t`, `f`.

`dbinom`, `pnorm`, `qpois`, `rexp` : les trente-six fonctions du cours se déduisent de deux listes. **Il n'y a rien d'autre à mémoriser.**

Sur R

```r
factorial(5)          # 120
choose(10, 3)         # 120  : le comité du chapitre 5
choose(49, 5)         # 1906884 : les grilles de loterie
prod(12:10)           # 1320 : l'arrangement A(12,3)
```

Trois remarques

`choose(n, k)` est $\binom{n}{k}$; il n'existe pas de fonction pour A_n^k, mais `prod(n:(n-k+1))` la donne.

`factorial(170)` est le dernier calcul possible : au-delà, R renvoie `Inf`. **Ne calculez jamais les factorielles séparément.**

```r
n <- 200 ; p <- 0.04

dbinom(8, n, p)          # 0.1424619  : exactement 8 défauts
1 - pbinom(13, n, p)     # 0.0312124  : au moins 14 défauts
qbinom(0.99, n, p)       # 15         : le quantile à 99 %
n*p                      # 8          : E(X)
sqrt(n*p*(1-p))          # 2.771281   : sigma(X)
```

Les cinq chiffres du chapitre 15

Le capital réglementaire s'écrit alors en une ligne :

```r
(qbinom(0.99, n, p) - n*p) * 100000   # 700000 dirhams
```

Poisson et l'approximation

```r
dpois(8, lambda = 8)        # 0.1395865  contre 0.1424619
ppois(14, lambda = 8)       # 0.9827430  contre 0.9847500
```

Le contrôle par une seconde méthode

Deux calculs indépendants qui s'accordent à trois millièmes près : le résultat est fiable.

C'est la vérification la moins coûteuse qui soit, et elle prend une ligne.

```r
pnorm(1.25)                       # 0.8943502  la table du chapitre 18
pnorm(9000, mean = 7158, sd = 1470)   # 0.8949083
1 - pnorm(9000, 7158, 1470)       # 0.1050917  les salaires élevés

qnorm(0.975)                      # 1.959964
qnorm(0.99)                       # 2.326348
```

La table entière tient dans une fonction

`pnorm(z)` est la table. Elle accepte directement m et σ, ce qui dispense même de standardiser.

`qnorm` fait la lecture inverse — et donne le $1{,}96$ de tous les intervalles de confiance.

Sur Python

Le module scipy.stats

```
from scipy import stats
from math import comb, factorial

comb(10, 3)                    # 120
factorial(5)                   # 120

n, p = 200, 0.04
stats.binom.pmf(8, n, p)       # 0.14246191  -> le d de R
1 - stats.binom.cdf(13, n, p)  # 0.03121238  -> le p de R
stats.binom.ppf(0.99, n, p)    # 15.0        -> le q de R
stats.binom.rvs(n, p, size=5)  # simulation  -> le r de R
```

Le module `scipy.stats` (suite)

La même logique, d'autres noms

R	Python	Rôle
`dbinom`	`binom.pmf`	probabilité d'une valeur
`pbinom`	`binom.cdf`	fonction de répartition
`qbinom`	`binom.ppf`	quantile
`rbinom`	`binom.rvs`	tirage

```
stats.norm.cdf(1.25)                    # 0.8943502
stats.norm.cdf(9000, loc=7158, scale=1470)   # 0.8949083
stats.norm.ppf(0.99)                    # 2.3263479

stats.expon.cdf(3, scale=1/0.2)         # 0.4511884
stats.poisson.pmf(8, mu=8)              # 0.1395865
```

Une seule précaution, mais elle compte

Python attend `scale`, c'est-à-dire l'**écart-type** pour la normale et $1/\lambda$ pour l'exponentielle.

Écrire `scale=0.2` au lieu de `scale=1/0.2` donne un résultat parfaitement plausible et parfaitement faux. **C'est l'erreur la plus fréquente du passage de R à Python.**

Vérifier les deux théorèmes

La loi des grands nombres

```r
set.seed(2024)
lancers <- sample(1:6, 100000, replace = TRUE)
moyennes <- cumsum(lancers) / seq_along(lancers)

moyennes[10]      # 3.900000
moyennes[1000]    # 3.472000
moyennes[100000]  # 3.499870

plot(moyennes, log = "x", type = "l")
abline(h = 3.5, col = "red", lwd = 2)
```

La figure du chapitre 20, reproduite en cinq lignes

Sur dix lancers, la moyenne vaut $3{,}9$; sur cent mille, $3{,}4999$.

Changez la graine et refaites : la trajectoire sera différente, la limite identique. C'est exactement ce que le théorème affirme.

```r
set.seed(2024)
n <- 30
moyennes <- replicate(10000, mean(sample(1:6, n, replace = TRUE)))

mean(moyennes)   # 3.499  -> m = 3.5
sd(moyennes)     # 0.312  -> sigma/sqrt(n) = 1.708/sqrt(30) = 0.312

hist(moyennes, breaks = 40, freq = FALSE)
curve(dnorm(x, 3.5, 1.708/sqrt(30)), add = TRUE, col = "red", lwd = 2)
```

Trois vérifications d'un coup

1. La moyenne des moyennes vaut bien $m = 3{,}5$.
2. Leur écart-type vaut bien $\sigma/\sqrt{n} = 0{,}312$.
3. L'histogramme épouse la courbe normale — alors que le dé est parfaitement plat.

Le même en Python

```python
import numpy as np
rng = np.random.default_rng(2024)

moyennes = rng.integers(1, 7, size=(10000, 30)).mean(axis=1)

moyennes.mean()   # 3.4995
moyennes.std()    # 0.3119
```

La simulation comme outil de gestion

Ce que nous venons de faire s'appelle une simulation de **Monte-Carlo**. Trois lignes suffisent.

Quand une loi est trop compliquée à calculer — un portefeuille aux actifs corrélés, un projet aux flux incertains — **on ne calcule plus : on simule dix mille scénarios et l'on lit le quantile**. C'est la méthode dominante en finance de marché.

Ce qu’il faut retenir

Le résumé du chapitre

Six points

1. Quatre préfixes en R — **d**, **p**, **q**, **r** — et le nom de la loi : trente-six fonctions en deux listes.
2. En Python : `pmf`/`pdf`, `cdf`, `ppf`, `rvs` dans `scipy.stats`.
3. `pnorm` et `norm.cdf` remplacent la table ; `qnorm` en fait la lecture inverse.
4. Attention au `scale` de Python : écart-type pour la normale, $1/\lambda$ pour l'exponentielle.
5. Cinq lignes reproduisent la loi des grands nombres, cinq autres le théorème central limite.
6. Quand le calcul devient impossible, la **simulation** donne la réponse.

La fin du cours

Nous sommes partis d'un dé et de six fréquences. Nous arrivons au capital réglementaire d'une banque et à la marge d'erreur d'un sondage.

Rien de nouveau n'a été inventé en chemin : la moyenne est devenue espérance, la fréquence probabilité, le tableau de contingence loi jointe. **Le calcul des probabilités est bien l'autre face de la statistique descriptive** — et il ouvre maintenant sur la statistique inférentielle, où l'on remontera de l'échantillon vers la population.