Échantillonnage

Chapitre 15 : Mise en œuvre sur R et Python

Jaouad Madkour

30 octobre 2026

Plan du chapitre

Tirer un échantillon **2**
Deux langages, les mêmes données . . . 2
Même graine, deux échantillons . . . 2
Partager la liste, pas la graine . . . 3
Strates et grappes avec sampling . . . 3
Deux pièges de sampling . . . 4

Le sondage aléatoire simple sous survey **4**
Déclarer le plan . . . 4
Moyenne, total, proportion . . . 5
L'intervalle de confiance . . . 5
Le même calcul en Python . . . 6

Strates et grappes sous survey **6**
Déclarer des strates . . . 6
Les strates en Python . . . 7
Cinq districts entiers . . . 7
Les grappes en Python . . . 8

Le ratio, la post-stratification et le calage **8**
L'estimateur par le ratio . . . 8
La post-stratification . . . 9
Le calage linéaire . . . 9
Le calage sur marges . . . 9
Redresser en Python . . . 10

Comparer les résultats **11**
Cinq estimations, un seul échantillon . . . 11
Des poids proches de dix . . . 11
Le logiciel et les formules du cours . . . 12
Correspondance entre R et Python . . . 13

Les pièges **13**
Les poids et la taille de la population . . . 13

Une variance estimée sous un mauvais plan . . . 14
Ce que coûte le mauvais plan . . . 14
Avant de publier . . . 15

S'entraîner **16**
Exercice 1 : lire une sortie . . . 16
Exercice 1 : solution . . . 16
Exercice 2 : oublier le fpc . . . 16
Exercice 2 : solution . . . 17
Exercice 3 : traduire en Python . . . 17
Exercice 3 : solution . . . 17

Ce qu'il faut retenir **18**
Le résumé du chapitre . . . 18

Tirer un échantillon

Deux langages, les mêmes données

Le matériel du chapitre

- **R** 4.3.3 : `sampling` 2.10 pour **tirer**, `survey` 4.2.1 pour **estimer** ; on déclare le plan, il en déduit poids, estimations et erreurs-types.
- **Python** : `numpy` et `pandas` pour tout écrire **à la main** ; `samplics` 0.6.0 refait les calculs de `survey` (à l'import, cette version avertit qu'elle n'est plus maintenue).

Charger la population du cours

```r
library(sampling)              # tirer
library(survey)                # estimer
d <- read.csv("menages.csv")
```

```python
import numpy as np, pandas as pd
d = pd.read_csv("menages.csv")
```

Chaque sortie de ce chapitre est celle de ces programmes, **exécutés** sur `menages.csv`.

Même graine, deux échantillons

Tirer 200 ménages parmi 2 000

```r
set.seed(2026)                 # la graine
s <- sample(2000, 200)         # 200 numéros parmi 2 000
head(s)
mean(d$depense[s])
```

```
[1]  733 1657  993  294 1581 1647
[1] 7803
```

```python
rng = np.random.default_rng(2026)               # la graine
s = rng.choice(2000, 200, replace=False) + 1    # 1 à 2000
print(s[:6], d.depense[d.id.isin(s)].mean())
```

```
[ 200  769 1747 1821 1560  307] 7843.05
```

Deux générateurs différents

R emploie par défaut le générateur Mersenne-Twister, `numpy` le générateur PCG64 : la graine 2026 donne deux sondages aléatoires simples **différents**, de moyennes 7 803 et 7 843,05 dh. Python retrouve l'échantillon fil rouge du chapitre 3 ; R en tire un autre, tout aussi aléatoire.

Partager la liste, pas la graine

Trois échantillons définis sans hasard

```r
ids <- read.csv("data/echantillon_fil_rouge.csv")$id
e <- d[d$id %in% ids, ]                  # le fil rouge
e9 <- d[seq(9, 2000, by = 10), ]         # systématique, départ 9
g5 <- d[d$district %in% c(2, 9, 19, 32, 40), ]   # 5 districts
c(mean(e$depense), mean(e9$depense), nrow(g5))
```

```
[1] 7843.05 7922.95  215.00
```

```python
ids = pd.read_csv("data/echantillon_fil_rouge.csv").id
e = d[d.id.isin(ids)]                      # le fil rouge
e9 = d.iloc[8::10]                         # départ 9 : rang 8
g5 = d[d.district.isin([2, 9, 19, 32, 40])]
print(e.depense.mean(), e9.depense.mean(), len(g5))
```

```
7843.05 7922.95 215
```

La méthode du chapitre

Pour comparer deux logiciels, il faut le **même échantillon** : on partage la liste des unités tirées, pas la graine. Ces trois échantillons des chapitres 3, 8 et 9 servent partout.

Strates et grappes avec sampling

Les fonctions strata et cluster

```r
set.seed(2026)
st <- sampling::strata(d, "province", method = "srswor",
                       size = c(82, 56, 38, 24))
table(getdata(d, st)$province)
set.seed(2026)
cl <- sampling::cluster(d, "district", size = 5,
                        method = "srswor")
unique(cl$district)
```

```
    Al Hoceïma        Larache Tanger-Assilah        Tétouan
            24             38             82             56
```

```
[1] 27 29 33 38 45
```

Ce que contiennent les résultats, et en Python

`st` donne la strate et la probabilité d'inclusion (colonne `Prob` : 0,1) des ménages tirés ; `cl`, les 156 ménages des districts tirés. En Python, un générateur neuf de graine 2026 tire les districts 2, 9, 19, 32 et 40 (chapitre 9) ; `groupby().sample(frac=0.1)` tire 10 % par province.

Deux pièges de sampling

Une fonction masquée

```
r1 <- try(strata(d, "province", size = c(82, 56, 38, 24)))
```

```
Error in strata(d, "province", size = c(82, 56, 38, 24)) :
  all arguments must be the same length
```

`survey` charge `survival`, dont `strata` et `cluster` **masquent** celles de `sampling` (R l'annonce au chargement) : écrire `sampling::strata`.

Des tailles attribuées dans l'ordre du fichier

```
set.seed(1); dm <- d[sample(2000), ]     # fichier mélangé
st2 <- sampling::strata(dm, "province", method = "srswor",
                        size = c(82, 56, 38, 24))
table(getdata(dm, st2)$province)
```

```
    Al Hoceïma        Larache Tanger-Assilah        Tétouan
            56             24             38             82
```

Les tailles suivent **l'ordre d'apparition** des strates : Al Hoceïma reçoit 56 ménages, sans message. Trier d'abord par strate.

Le sondage aléatoire simple sous survey

Déclarer le plan

La fonction svydesign

- `ids` : les unités tirées au premier degré ; `~1` : les ménages eux-mêmes, sans grappes.
- `fpc` : la taille de la population N, ou le taux f. `survey` en déduit les poids $d_k = N/n$ et le facteur $1 - f$.
- `strata`, `weights` ou `probs` : les strates, les poids d_k ou les π_k, quand il y en a.
- `data` : le fichier des **seules unités tirées**.

Le fil rouge

```
e$N <- 2000                      # taille de la population
p <- svydesign(ids = ~1, fpc = ~N, data = e)
unique(weights(p))               # d_k = N / n
degf(p)                          # n - 1
```

```
[1] 10
```

```
[1] 199
```

Chaque ménage tiré en représente 10 (chapitre 3) ; la variance sera estimée avec $n - 1 = 199$ degrés de liberté.

Moyenne, total, proportion

Les fonctions svymean et svytotal

```
svymean(~depense, p)
svytotal(~depense, p)
svymean(~internet, p)
```

```
         mean     SE
depense 7843 256.87
            total     SE
depense 15686100 513736
          mean     SE
internet 0.67 0.0316
```

Les formules des chapitres 3 et 4

- $\sqrt{0{,}9 \times 3\,829{,}16^2/200} = 256{,}87$: c'est $\sqrt{(1-f)\,s^2/n}$. R arrondit l'affichage de la moyenne : `coef()` donne 7 843,05.
- Total : $2\,000 \times 7\,843{,}05 = 15\,686\,100$ dh, d'erreur-type $2\,000 \times 256{,}87 = 513\,736$.
- Proportion : $\sqrt{0{,}9 \times 0{,}67 \times 0{,}33/199} = 0{,}0316$.

L'intervalle de confiance

confint, avec ou sans degrés de liberté

```
m <- svymean(~depense, p)
confint(m)                      # loi normale : 1,96
confint(m, df = degf(p))        # loi de Student : 199 ddl
cv(m)                           # SE / moyenne
```

```
            2.5 %   97.5 %
depense 7339.598 8346.502
            2.5 %   97.5 %
depense 7336.518 8349.582
            depense
depense 0.03275101
```

Normale ou Student

Par défaut, `confint` prend 1,96 : c'est l'intervalle [7 339,6 ; 8 346,5] du chapitre 3, de coefficient de variation 3,28 %. Avec `df = degf(p)`, il prend $t = 1{,}972$, quantile de la loi de Student à 199 degrés de liberté (Statistique inférentielle, chapitre 4). Ici l'écart est négligeable ; avec cinq grappes, il ne le sera plus.

Le même calcul en Python

À la main, avec numpy et pandas

```
n, N = len(e), 2000
ybar, s = e.depense.mean(), e.depense.std()    # s : en n - 1
se = np.sqrt((1 - n / N) * s**2 / n)
print(round(ybar, 2), round(se, 2), N * ybar, round(N * se))
print(round(ybar - 1.96 * se, 1), round(ybar + 1.96 * se, 1))
```

```
7843.05 256.87 15686100.0 513736
7339.6 8346.5
```

Avec samplics, qui donne d'office l'intervalle de Student

```
from samplics import TaylorEstimator, PopParam
m = TaylorEstimator(PopParam.mean)
m.estimate(y=e.depense, samp_weight=np.full(n, N / n),
           fpc=1 - n / N)                  # samplics : 1 - f
print(m)                                   # dernières lignes :
```

```
   MEAN         SE         LCI         UCI       CV
7843.05 256.867802 7336.517857 8349.582143 0.032751
```

Strates et grappes sous survey

Déclarer des strates

L'échantillon systématique, déclaré stratifié

```
Nh <- table(d$province)                # 820, 560, 380, 240
e9$Nh <- as.numeric(Nh[e9$province])
ps <- svydesign(ids = ~1, strata = ~province,
                fpc = ~Nh, data = e9)
svymean(~depense, ps, deff = TRUE)
degf(ps)                               # n - H
e9$N <- 2000                           # le même, comme un SAS
svymean(~depense, svydesign(ids = ~1, fpc = ~N, data = e9))
```

```
           mean      SE   DEff
depense 7922.95  260.91 0.9476
[1] 196
        mean     SE
depense 7923 268.02
```

Une stratification implicite rendue explicite

Rangé par province, le fichier donne au pas de 10 exactement 82, 56, 38 et 24 ménages (chapitre 8). Effet de plan **estimé** : 0,948, pour 0,954 en théorie (chapitre 6).

Les strates en Python

La formule du chapitre 6, écrite à la main

```
Nh = d.province.value_counts()              # 820, 560, 380, 240
g = e9.groupby("province").depense
nh, W = g.size(), Nh / 2000
yst = (W * g.mean()).sum()
V = (W**2 * (1 - nh / Nh) * g.var() / nh).sum()
print(round(yst, 2), round(np.sqrt(V), 2))
```

```
7922.95 260.91
```

Ce que fait chaque ligne

$$\overline{y}_{st} = \sum_h W_h \, \overline{y}_h \qquad \widehat{V}(\overline{y}_{st}) = \sum_h W_h^2 \, (1 - f_h) \, \frac{s_h^2}{n_h}$$

`groupby` calcule n_h, $\overline{y}_h$ et s_h^2 (`var()` divise par $n_h - 1$) ; `pandas` aligne les séries sur le nom des provinces. Les degrés de liberté sont $n - H = 200 - 4 = 196$: une moyenne estimée par strate.

Cinq districts entiers

ids nomme les grappes

```
g5$NI <- 50                                 # districts de la population
pg <- svydesign(ids = ~district, fpc = ~NI, data = g5)
m5 <- svymean(~depense, pg, deff = TRUE)
m5
confint(m5)                                 # 1,96 : trop étroit
confint(m5, df = degf(pg))                  # Student à 4 ddl
```

```
           mean      SE   DEff
depense 8320.93  529.49 4.5622
           2.5 %   97.5 %
depense 7283.152 9358.709
           2.5 %   97.5 %
depense 6850.834 9791.026
```

Ce que calcule survey

`svymean` donne le **ratio** du chapitre 9, $1\,789\,000/215 = 8\,320{,}93$ dh ; `svytotal`, l'estimateur de Horvitz-Thompson, 17 890 000 dh. `degf(pg)` vaut $n_I - 1 = 4$: la variance repose sur **cinq** totaux de districts, d'où $t = 2{,}776$.

Les grappes en Python

Le ratio et ses résidus, district par district

```
from scipy.stats import t
G = g5.groupby("district").depense.agg(["size", "sum"])
r = G["sum"].sum() / G["size"].sum()       # le ratio
res = G["sum"] - r * G["size"]             # résidus e_i
Nc = 50 / 5 * G["size"].sum()              # N estimé : 2 150
se = 50 * np.sqrt(0.9 * res.var() / 5) / Nc
q = t.ppf(0.975, 4)                        # 2,776
print(round(r, 2), round(se, 2), Nc)
print(round(r - q * se, 1), round(r + q * se, 1))
```

```
8320.93 529.49 2150.0
6850.8 9791.0
```

529 dh ici, 569 dh au chapitre 9

Le chapitre 9 divisait par $n_I\,\overline{M} = 200$ ménages attendus ; `survey` divise par le nombre **estimé** $\widehat{N} = 10 \times 215 = 2\,150$: $569{,}2 \times 200/215 = 529{,}5$. Deux linéarisations du même ratio ; la seconde n'exige pas de connaître N.

Le ratio, la post-stratification et le calage

L'estimateur par le ratio

svyratio, puis predict

```
r <- svyratio(~depense, ~taille, p)
c(coef(r), SE(r))                          # dépense par personne
pr <- predict(r, total = 7859)             # X = 7 859 personnes
c(pr$total, pr$se) / 2000                  # par ménage
```

```
depense/taille depense/taille
    2061.24836       53.05089
[1] 8099.6754  208.4635
```

Les nombres du chapitre 12

$\widehat{R} = 2\,061{,}25$ dh par personne, d'erreur-type 53,05 ; estimation de la dépense moyenne : $3{,}9295 \times 2\,061{,}25 = 8\,099{,}68$ dh. L'erreur-type, 208,46, multiplie celle de $\widehat{R}$ par $\overline{X} = 3{,}9295$; le chapitre 12 la multipliait par $\overline{x} = 3{,}805$, d'où 201,86. Deux approximations, à 3 % l'une de l'autre.

La post-stratification

postStratify et les effectifs connus

```
pm <- data.frame(milieu = c("rural", "urbain"),
                 Freq = c(562, 1438))  # effectifs connus
pps <- postStratify(p, ~milieu, pm)
svymean(~depense, pps)
unique(weights(pps))
```

```
          mean     SE
depense 7793.1 252.43
[1]  9.460526 11.708333
```

Les poids du chapitre 13

Les poids deviennent $1\,438/152 = 9{,}461$ pour les urbains et $562/48 = 11{,}708$ pour les ruraux ; l'estimation est celle du chapitre 13, 7 793,07 dh. Le nom des colonnes compte : `milieu` doit s'écrire comme dans `e`, et l'effectif s'appeler `Freq`.

Le calage linéaire

calibrate sur le nombre de ménages et de personnes

```
tot <- c(`(Intercept)` = 2000, taille = 7859)
pc <- calibrate(p, ~taille, tot)       # N et X connus
svymean(~depense, pc)
svytotal(~taille, pc)
range(weights(pc))
```

```
          mean     SE
depense 8024.7 197.67
       total SE
taille  7859  0
[1]  8.893807 12.443090
```

L'estimateur par la régression

La formule `~taille` contient une constante : on cale sur $\sum_S w_k = 2\,000$ **et** $\sum_S w_k\, x_k = 7\,859$. Le total des personnes est reproduit exactement, d'où son erreur-type nulle. Poids de 8,89 à 12,44 et estimation 8 024,74 dh : ceux du chapitre 13.

Le calage sur marges

rake, milieu et province

```
pp <- data.frame(province = c("Tanger-Assilah", "Tétouan",
                              "Larache", "Al Hoceïma"),
                 Freq = c(820, 560, 380, 240))
pk <- rake(p, list(~milieu, ~province), list(pm, pp))
svymean(~depense, pk)
range(weights(pk))
```

```
          mean     SE
depense 7852.1 245.99
[1]  7.484064 15.364057
```

L'ajustement proportionnel itératif

`rake` répète les post-stratifications sur chaque marge, comme au chapitre 13 : poids de 7,48 à 15,36, estimation 7 852,1 dh. Il s'arrête dès que chaque marge est juste à un ménage près (option `epsilon`, qui vaut 1) : 561,93 ruraux pour 562. Une tolérance plus fine donne 7 852,13 dh.

Redresser en Python

Ratio, post-stratification et régression à la main

```
x, y = e.taille, e.depense
R = y.sum() / x.sum()                          # ratio
Nm = pd.Series({"rural": 562, "urbain": 1438})
post = (Nm / 2000 * y.groupby(e.milieu).mean()).sum()
b = np.cov(x, y)[0, 1] / x.var()               # pente
reg = y.mean() + b * (7859 / 2000 - x.mean())
print(round(R, 2), round(post, 2), round(reg, 2))
```

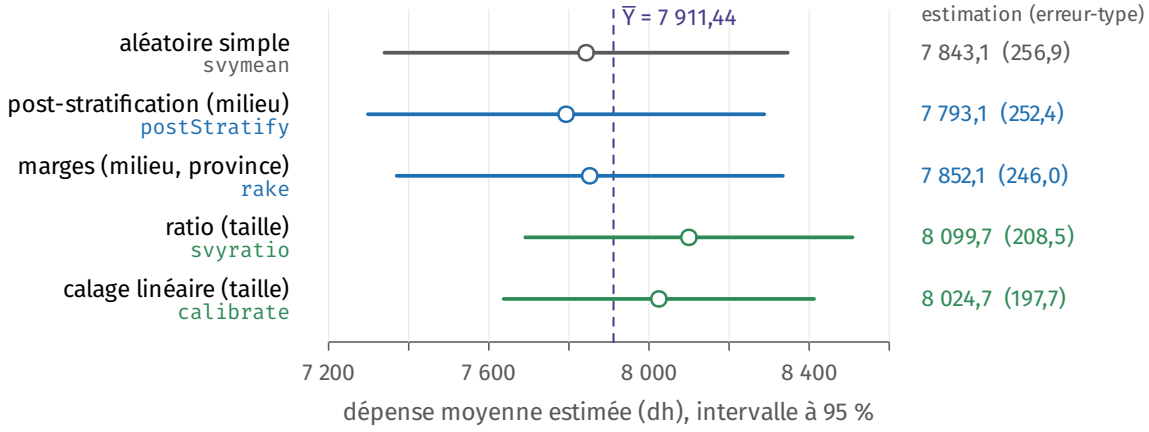
```
2061.25 7793.07 8024.74
```

Et les poids calés

Le calage linéaire **est** l'estimateur par la régression (chapitre 13) : `reg` vaut l'estimation de `calibrate`. Pour obtenir les poids, `samplics` fournit `SampleWeight()`, avec les méthodes `poststratify`, `calibrate` et `rake` : son calage sur le milieu et la province donne 7 852,13 dh. La boucle de l'ajustement itératif s'écrit aussi en quelques lignes (feuille d'exercices).

Comparer les résultats

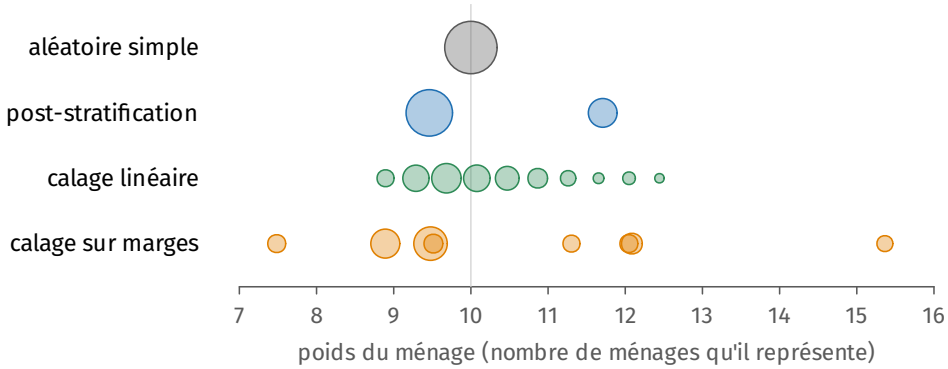
Cinq estimations, un seul échantillon



Ce que montre la figure

Les cinq intervalles, calculés par `survey` sur le fil rouge, contiennent $\overline{Y}$. Le milieu et la province changent peu l'estimation et sa précision ; la taille du ménage réduit l'erreur-type de 19 à 23 %, mais, sur **cet** échantillon, éloigne l'estimation de $\overline{Y}$ (chapitres 12 et 13).

Des poids proches de dix



Le calage retouche les poids sans les bouleverser

Chaque cercle est un poids, de surface proportionnelle au nombre de ménages qui le portent. Tous les jeux de poids somment à $N = 2\,000$. Le calage linéaire les ordonne selon la taille ; le calage sur marges les écarte le plus (7,48 à 15,36) : il corrige à la fois Larache, sous-représentée, et Al Hoceïma, sur-représentée.

Le logiciel et les formules du cours

Mêmes estimations, erreurs-types voisines

Erreur-type	Cours	survey	Écart
Ratio (ch. 12)	201,86	208,46	facteur $\overline{X}$
Post-strat. (ch. 13)	253,1	252,43	linéarisation
Calage (ch. 13)	189,1	197,67	résidus pondérés
Grappes (ch. 9)	569	529,49	$\widehat{N}$, non N

Plusieurs approximations d'une même variance

Les estimations coïncident au centime. Les erreurs-types diffèrent de 0,3 à 7,5 % : ce sont des approximations, toutes valables quand n est grand, de la variance d'un estimateur non linéaire. Pour le calage, `survey` multiplie chaque résidu par son poids calé et divise par $n-1$, non $n-2$. On indique toujours le logiciel employé.

Démonstration : Un seul calcul pour les erreurs-types du logiciel

La forme commune. Pour chacun des estimateurs du chapitre, `survey` calcule l'erreur-type d'une somme $\sum_{k\in s} z_k$, avec $z_k = w_k\, e_k/N$: w_k est le poids final du ménage k, et e_k un résidu qui dépend de l'estimateur : y_k pour le sondage simple, $y_k - \widehat{R}\, x_k$ pour le ratio, $y_k - \overline{y}_h$ pour la post-stratification, $y_k - a - b\, x_k$ pour le calage.

Le calcul. Posons $u_k = n\, z_k$: la somme $\sum_k z_k$ est la moyenne $\overline{u}$ des u_k. Estimée comme dans un sondage simple, sa variance est $(1-f)\, s_u^2/n$, et $s_u^2 = n^2 s_z^2$:

$$\widehat{V} = (1-f)\, n\, s_z^2 = (1-f)\, \frac{n}{n-1} \sum_{k\in s} (z_k - \overline{z})^2.$$

Le sondage simple. Avec $w_k = N/n$ et $e_k = y_k$, on a $z_k = y_k/n$ et $\sum_k (z_k - \overline{z})^2 = (n-1)\, s^2/n^2$: on retrouve $\widehat{V} = (1-f)\, s^2/n$. Pour une variable 0/1, $\sum_k y_k^2 = \sum_k y_k = n\, \hat{p}$, donc $s^2 = \frac{n}{n-1}\, \hat{p}\, (1-\hat{p})$ et l'erreur-type d'une proportion est $\sqrt{(1-f)\, \hat{p}\, (1-\hat{p})/(n-1)}$. Un total vaut N fois la moyenne, comme son erreur-type.

Le ratio. Avec $w_k = \frac{N}{n}\, \frac{\overline{X}}{\overline{x}}$, on a $z_k = \frac{\overline{X}}{\overline{x}}\, \frac{e_k}{n}$ et

$$\widehat{V} = \left(\frac{\overline{X}}{\overline{x}}\right)^2 (1-f)\, \frac{s_e^2}{n}.$$

Le chapitre 12 donne $(1-f)\, s_e^2/n$: la même formule, sans le carré du facteur $\overline{X}/\overline{x}$.

La post-stratification. Avec $w_k = N_h/n_h$ et $e_k = y_k - \overline{y}_h$, les z_k somment à zéro dans chaque strate, et

$$\widehat{V} = (1-f)\, \frac{n}{n-1} \sum_h W_h^2\, \frac{n_h - 1}{n_h^2}\, s_h^2,$$

alors que le chapitre 13 écrit $\sum_h W_h^2\, (1 - n_h/N_h)\, s_h^2/n_h$: le facteur de correction y est compté strate par strate.

Le calage. Les poids calés ne sont plus égaux à $N/n = 10$: ils multiplient chaque résidu dans z_k, et le diviseur est $n-1$. Le chapitre 13 garde le poids 10 et divise par $n-2$.

En Python. Le $n-1$ est celui de s^2 : `np.std(y)` divise par n et donne $s\sqrt{(n-1)/n}$; pour les 2 000 ménages, $3\,984{,}29\sqrt{1\,999/2\,000} = 3\,983{,}29$.

Ce que cela dit. Sur le fil rouge, la formule redonne les quatre nombres de `survey` : 256,87 pour la moyenne simple ; 201,86 × 1,0327 = 208,46 pour le ratio ($\overline{X}/\overline{x} = 3{,}9295/3{,}805$) ; 252,43 pour la post-stratification, contre 253,1 au chapitre 13 ; 197,67 pour le calage, dont les poids vont de 8,89 à 12,44, contre 189,1. Les écarts avec le cours sont de +3,3 %, −0,3 % et +4,5 % : trois façons voisines d'estimer une même variance. Un total est multiplié par N : 2 000 × 256,87 = 513 736 ; une proportion de 0,67 a pour erreur-type $\sqrt{0{,}9 \times 0{,}67 \times 0{,}33/199} = 0{,}0316$.

■

Correspondance entre R et Python

Les mêmes opérations dans les deux langages

Opération	R	Python
Écart-type en $n-1$	`sd(y)`	`y.std()`, `np.std(y, ddof=1)`
Sondage simple	`sample(N, n)`	`rng.choice(N, n, ...)`
Tirage stratifié	`sampling::strata`	`groupby().sample()`
Déclarer le plan	`svydesign()`	poids et formules
Correction `fpc`	N ou f	samplics : $1-f$
Moyenne, total	`svymean`, `svytotal`	`np.average`, `sum`
Domaines	`svyby`	`groupby`, `domain=`
Calage	`calibrate`, `rake`	`SampleWeight()`
Quantile de Student	`qt(0.975, df)`	`t.ppf(0.975, df)`

Attention : `np.std(y)` divise par n, comme σ au chapitre 1 (3 983,29 au lieu de 3 984,29).

Les pièges

Les poids et la taille de la population

Deux déclarations fausses, sans message d'erreur

```r
e$pi <- 0.1; e$n <- 200
p1 <- svydesign(ids = ~1, weights = ~pi, data = e)
p3 <- svydesign(ids = ~1, fpc = ~n, data = e)
svytotal(~depense, p1)                # poids = pi
svymean(~depense, p3)                 # fpc = n
```

```
         total     SE
depense 156861 5415.2
        mean SE
depense 7843  0
```

Lire ce que l'on a déclaré

- `weights` attend les poids $d_k = 1/\pi_k$, et `probs` les π_k. Avec π_k comme poids, le total est **cent fois** trop petit : 156 861 dh au lieu de 15 686 100.
- `fpc` attend N. Avec n, `survey` croit à un recensement ($f = 1$) : erreur-type **nulle**.

Les moyennes restent justes : seul un coup d'œil aux totaux et aux erreurs-types révèle l'erreur.

Une variance estimée sous un mauvais plan

Les cinq districts déclarés comme un SAS de ménages

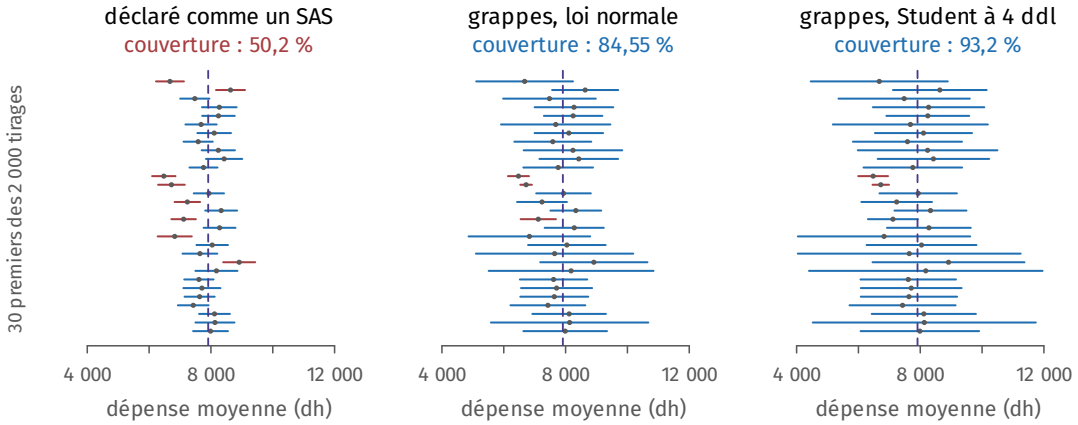
```
g5$N <- 2000                              # 215 ménages « au hasard »
faux <- svydesign(ids = ~1, fpc = ~N, data = g5)
svymean(~depense, faux)
confint(svymean(~depense, faux))          # comme un SAS
confint(m5, df = degf(pg))                # comme des grappes
```

```
          mean     SE
depense 8320.9 246.86
            2.5 %   97.5 %
depense 7837.093 8804.768
            2.5 %   97.5 %
depense 6850.834 9791.026
```

215 ménages, mais cinq grappes

La même estimation, deux précisions. Déclaré comme un SAS, l'échantillon se croit fait de 215 ménages indépendants : erreur-type 246,86, intervalle qui **manque** $\overline{Y}$ = 7 911,44. Les voisins se ressemblent ($\rho = 0{,}162$, chapitre 9) : l'information vient de cinq districts.

Ce que coûte le mauvais plan



Deux mille tirages de cinq districts, sous R

Déclaré comme un SAS, l'intervalle à 95 % ne contient $\overline{Y}$ qu'**une fois sur deux** (50,2 %). Avec les grappes et 1,96 : 84,55 % ; avec la loi de Student à 4 degrés de liberté : 93,2 %. Lar-

geurs moyennes : 1 020 ; 2 627 et 3 721 dh.

Démonstration : Ce que calcule le logiciel avec une déclaration fausse

Sans fpc. Le facteur $1-f$ disparaît de $\widehat{V} = (1-f)\,s^2/n$: l'erreur-type devient $s/\sqrt{n}$ au lieu de $\sqrt{1-f}\;s/\sqrt{n}$. Le rapport des deux est $\sqrt{1-f}$, et l'erreur-type est surestimée de $1/\sqrt{1-f} - 1$.

Un fpc égal à n. `survey` lit $N = n$, donc $f = n/n = 1$ et $1 - f = 0$: la variance estimée est nulle, quel que soit s^2.

Des poids égaux à π. Le total estimé est $\sum_k w_k\, y_k$. Avec $w_k = \pi$ au lieu de $1/\pi$ pour tous les ménages, il vaut $\pi \sum_k y_k$ au lieu de $\dfrac{1}{\pi}\sum_k y_k$: le rapport est π^2. La moyenne $\sum_k w_k y_k \big/ \sum_k w_k$ ne change pas quand tous les poids sont multipliés par une même constante.

Un plan faux. Déclarés comme un sondage simple de 215 ménages, les ménages donnent l'erreur-type $\sqrt{(1 - 215/2\,000)\,S^2/215}$. Or l'estimateur par le ratio sur cinq districts a une vraie erreur-type $\sqrt{V}$ bien plus grande (chapitre 9). L'intervalle $\overline{y} \pm 1{,}96\,\mathrm{SE}_{\text{déclaré}}$ contient $\overline{Y}$ avec la probabilité, en approximation normale,

$$2\,\Phi\Big(1{,}96\,\frac{\mathrm{SE}_{\text{déclaré}}}{\sqrt{V}}\Big) - 1.$$

Student. Avec cinq districts, la loi de Student à 4 degrés de liberté remplace 1,96 par 2,776 : la largeur de l'intervalle est multipliée par $2{,}776/1{,}96 = 1{,}416$.

Ce que cela dit. Sur le fil rouge, sans `fpc`, $\sqrt{0{,}9} = 0{,}9487 = 256{,}87/270{,}76$: surestimation de 5,4 % pour $f = 0{,}1$, mais $\sqrt{1 - 0{,}0019} = 0{,}999$ pour $f = 0{,}0019$. Avec des poids égaux à $\pi = 0{,}1$, le total vaut $0{,}1 \times 1\,568\,610 = 156\,861$ au lieu de $10 \times 1\,568\,610 = 15\,686\,100$, soit $0{,}1/10 = 1/100$ du bon total. Pour les cinq districts, un sondage simple de 215 ménages aurait une erreur-type de 256,7 (le logiciel a donné 246,86 sur l'échantillon tiré), la vraie est de 773,8 : le rapport 0,332 donne $2\,\Phi(0{,}650) - 1 = 0{,}484$, proche des 50,2 % simulés. Enfin $2\,627 \times 1{,}416 = 3\,721$ dh.

■

Avant de publier

Cinq vérifications

1. Le plan déclaré est **celui du tirage** : grappes dans `ids`, strates dans `strata`, N dans `fpc`.
2. La somme des poids vaut N ; après calage, les totaux connus sont reproduits.
3. `degf` vaut $n-1$, $n-H$ ou $n_I - 1$; l'intervalle se calcule avec `df = degf(p)`.
4. Un cas simple, le SAS, est refait à la main : mêmes nombres.
5. On conserve la liste des unités tirées, la graine et les versions des logiciels.

Le logiciel ne sait que ce qu'on lui dit

`survey` ignore comment l'échantillon a été tiré : il applique les formules du plan **déclaré**. Un plan faux donne des nombres faux, avec le même nombre de décimales que des nombres justes.

S'entraîner

Exercice 1 : lire une sortie

Les ménages connectés, échantillon stratifié

Sur l'échantillon systématique `e9`, déclaré stratifié par province (plan `ps`) :

```
svymean(~internet, ps, deff = TRUE)
svytotal(~internet, ps)
```

```
             mean       SE   DEff
internet 0.695000 0.029992 0.9383
          total     SE
internet   1390 59.984
```

1. Donner l'intervalle à 95 % de la proportion, puis du nombre de ménages connectés. Contiennent-ils $P = 0{,}674$ et 1 348 ?
2. Vérifier le lien entre les deux erreurs-types.
3. Interpréter `DEff` ; calculer la taille effective.

Exercice 1 : solution

Les intervalles

1. $0{,}695 \pm 1{,}96 \times 0{,}029992$, soit $[0{,}636\,;0{,}754]$; $1\,390 \pm 1{,}96 \times 59{,}984$, soit $[1\,272{,}4\,;1\,507{,}6]$ ménages. Les deux contiennent la vraie valeur.
2. Le total est N fois la proportion : $2\,000 \times 0{,}695 = 1\,390$ et $2\,000 \times 0{,}029992 = 59{,}984$.

L'effet de plan

3. La variance estimée est 6,2 % plus faible que celle d'un SAS de 200 ménages : ces 200 ménages en valent $200/0{,}9383 = 213$. C'est une **estimation** : l'effet de plan théorique de la province pour internet vaut 0,966 (chapitre 7).

Exercice 2 : oublier le fpc

Des poids, mais pas de taille de population

Un étudiant déclare le fil rouge avec ses poids, sans `fpc` :

```
e$d <- 10                                  # poids, sans fpc
p0 <- svydesign(ids = ~1, weights = ~d, data = e)
svymean(~depense, p0)
```

```
         mean     SE
depense 7843 270.76
```

1. Comparer à l'erreur-type de `p` (256,87) : calculer leur rapport. D'où vient-il ?

2. Retrouver 270,76 à partir de $s = 3\,829{,}16$. Quel plan `survey` a-t-il supposé ?
3. L'erreur est-elle grave ? Et pour l'enquête de 2022 sur le niveau de vie ($f = 0{,}0019$) ?

Exercice 2 : solution

Le facteur oublié

1. $256{,}87/270{,}76 = 0{,}9487 = \sqrt{0{,}9} = \sqrt{1-f}$: sans `fpc`, le facteur de correction de population finie disparaît.
2. $3\,829{,}16/\sqrt{200} = 270{,}76$: c'est $s/\sqrt{n}$, la formule du tirage **avec remise** (chapitre 4).

Une erreur prudente

3. L'erreur-type est surestimée de 5,4 % : l'intervalle est un peu trop large, du bon côté. Avec $f = 0{,}0019$, $\sqrt{1-f} = 0{,}999$: oublier `fpc` ne change rien, on peut l'omettre.

Exercice 3 : traduire en Python

La moyenne d'un domaine

R donne, pour les deux milieux du fil rouge :

```
svyby(~depense, ~milieu, p, svymean)
```

```
       milieu  depense       se
rural   rural 6916.667 458.4926
urbain urbain 8135.592 301.9070
```

Écrire en Python le calcul de la moyenne des ruraux et de son erreur-type, avec la formule du chapitre 12 : le domaine est un ratio, de résidus $e_k = \delta_k\,(y_k - \overline{y}_d)$, et

$$\widehat{V}(\overline{y}_d) = \frac{1-f}{n} \cdot \frac{s_e^2}{(n_d/n)^2}$$

Exercice 3 : solution

Le calcul, avec e, y, n et N déjà définis

```
rur = (e.milieu == "rural").values
yd = y[rur].mean()                              # moyenne du domaine
z = np.where(rur, y - yd, 0)                    # résidus e_k
se_d = np.sqrt((1 - n / N) * z.var(ddof=1) / n) / rur.mean()
print(rur.sum(), round(yd, 2), round(se_d, 2))
```

```
48 6916.67 458.49
```

Lecture

`rur.mean()` vaut $n_d/n = 48/200$. Les résidus sont nuls hors du domaine, mais ils comptent dans la variance : la taille du domaine est **aléatoire**. `samplics` donne le même résultat avec l'option `domain=e.milieu`.

Ce qu'il faut retenir

Le résumé du chapitre

Sept points

1. R : `sampling` tire, `survey` estime. Python : `numpy` et `pandas` à la main, ou `samplics`.
2. Même graine, autre générateur : pour comparer, on partage la **liste** des unités tirées.
3. `svydesign` déclare le plan : `ids` (grappes), `strata`, `fpc` égal à N.
4. `svymean`, `svytotal`, `svyratio`, `svyby` ; l'intervalle avec `df = degf(p)`.
5. `postStratify`, `calibrate` et `rake` donnent les poids calés du chapitre 13.
6. Les erreurs-types diffèrent un peu des formules du cours : autres approximations.
7. Poids égal à π_k, `fpc` égal à n, mauvais plan : des résultats faux, sans message.

La suite

Le cours s'achève. La Statistique inférentielle généralisera la démarche à des modèles ; devant des données d'enquête, la première question restera : **comment l'échantillon a-t-il été tiré ?**