

TP 2 – SY02

Probabilités

Corrigé

Les questions/sections marquées par un  sont des questions qui sont prévues pour être traitées en autonomie en dehors de la séance de TP.

1 Étude de peintres

Roger de Piles est un peintre et critique d'art français du 18ème siècle. Pour évaluer un peintre, de Piles a considéré quatre critères : composition, dessin, couleur, expression. Chacune de ces caractéristiques est notée sur 20. Plusieurs écoles artistiques ont été étudiés par de Piles : (A) renaissance, (B) maniériste, (C) italienne du 17ème siècle, (D) vénitienne, (E) lombarde, (F) 16ème siècle, (G) 17ème siècle, (H) française. Dans le logiciel R, on peut accéder aux notes de 54 peintres évalués par de Piles en chargeant tout d'abord la bibliothèque MASS :

```
library(MASS)
```

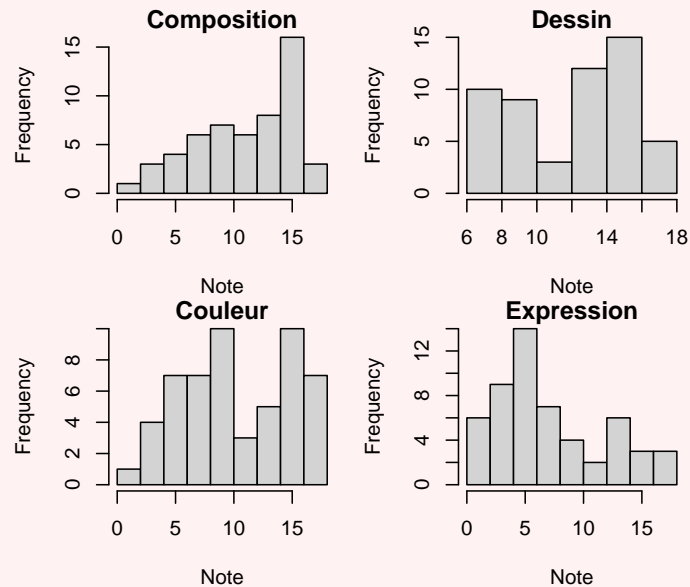
Les données sont alors accessibles à travers le `data.frame` `painters` :

```
head(painters)
```

	Composition	Drawing	Colour	Expression	School
Da Udine	10	8	16	3	A
Da Vinci	15	16	4	14	A
Del Piombo	8	13	16	7	A
Del Sarto	12	16	9	8	A
Fr. Penni	0	15	8	0	A
Guilio Romano	15	16	4	14	A

- ① En utilisant des histogrammes, visualiser la distribution des notes pour chaque critère.

```
par(mfrow = c(2, 2)) # Permet d'afficher plusieurs graphiques dans une seule fenêtre
hist(painters$Composition, main = "Composition", xlab = "Note")
hist(painters$Drawing, main = "Dessin", xlab = "Note")
hist(painters$Colour, main = "Couleur", xlab = "Note")
hist(painters$Expression, main = "Expression", xlab = "Note")
```



- ② Calculer dans un vecteur `moyenne` la moyenne des quatre notes de chaque peintre.

```
(moyenne <- (painters[, 1] + painters[, 2] + painters[, 3] + painters[, 4])/4)
[1] 9.25 12.25 11.00 11.25 5.75 12.25 9.25 11.00 7.50 16.25 9.75 11.00
[13] 9.25 11.50 11.50 10.00 11.25 12.00 7.00 10.00 8.00 13.75 7.75 6.00
[25] 9.75 8.25 10.25 6.75 11.00 12.25 12.75 11.00 11.00 7.00 13.25 14.50
[37] 10.50 10.50 14.50 9.00 12.00 7.75 6.00 10.25 10.00 11.75 12.50 16.25
[49] 11.50 13.75 7.50 14.00 12.25 13.25
```

- ③ Calculer la moyenne empirique, les variances et écarts types empiriques corrigés et non-corrigés de ce vecteur `moyenne` en n'utilisant que la fonction `sum`.

```
n <- length(moyenne)
(x_ <- sum(moyenne)/n)                                     # moyenne empirique
[1] 10.65741
(s2 <- sum((moyenne - x_)^2)/n)                             # variance empirique
[1] 6.171982
(s <- sqrt(s2))                                             # écart-type empirique
[1] 2.484347
(s2_ <- sum((moyenne - x_)^2)/(n - 1))                     # variance empirique corrigée
[1] 6.288435
(s_ <- sqrt(s2_))                                           # écart-type empirique corrigé
[1] 2.507675
```

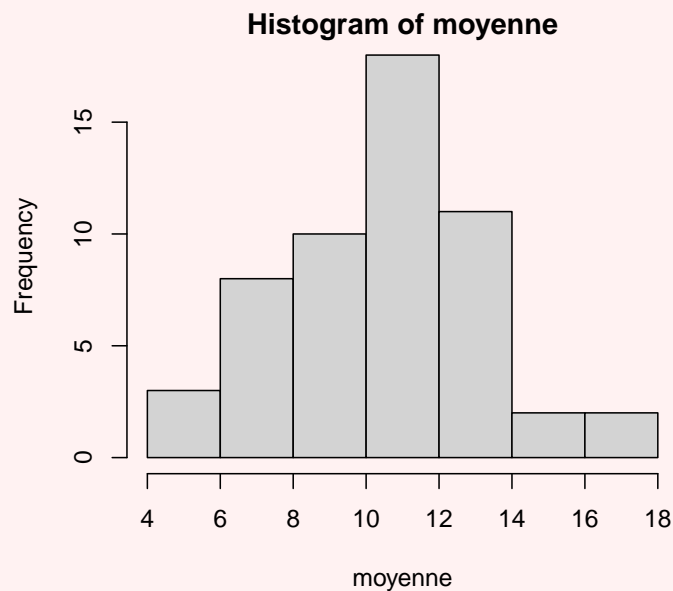
- ④ Retrouver les valeurs précédentes à l'aide des fonctions `mean`, `var` et `sd`.

```
mean(moyenne)                                              # moyenne empirique
[1] 10.65741
var(moyenne)                                              # variance empirique corrigée
[1] 6.288435
sd(moyenne)                                              # écart-type empirique corrigé
[1] 2.507675
```

```
var(moyenne) * ((n - 1)/n)           # variance empirique
[1] 6.171982
sd(moyenne) * sqrt((n - 1)/n)       # écart-type empirique
[1] 2.484347
```

- ⑤ Tracer l'histogramme du vecteur `moyenne` et commenter le résultat.

```
hist(moyenne)
```



Alors que la distribution des notes pour chaque critère n'est pas gaussienne, la moyenne de ces notes semble plus en accord avec une loi normale.

2 Calcul de probabilités

Plusieurs lois de probabilités sont incluses par défaut dans R. Parmi les plus communes, on peut citer :

- la loi uniforme : `unif`;
- la loi de Poisson : `pois`;
- la loi exponentielle : `exp`;
- la loi binomiale : `binom`;
- la loi normale : `norm`;
- la loi de Student : `t`;
- la loi du χ^2 : `chisq`;
- la loi de Fisher : `f`.

Pour chacune de ces lois, il est possible d'accéder à la fonction de densité en ajoutant le préfixe `d`, à la fonction de répartition avec `p`, aux fractiles avec `q` et à un générateur de nombre aléatoire en utilisant le préfixe `r`. À titre d'exemple, étudions la loi uniforme sur l'intervalle $[2, 5]$ notée $\mathcal{U}([2, 5])$. La valeur de sa fonction de densité $f(x)$ en un point x peut être calculée à l'aide de la fonction `dunif`.

```
dunif(1, min = 2, max = 5)
[1] 0
dunif(2, min = 2, max = 5)
[1] 0.3333333
dunif(4, min = 2, max = 5)
[1] 0.3333333
dunif(6, min = 2, max = 5)
[1] 0
```

Les arguments `min` et `max` permettent de spécifier l'intervalle $[2, 5]$. Ces paramètres sont propres à la loi uniforme. Pour chaque loi disponible dans R, tous les paramètres associés sont listés et expliqués dans l'aide.

Soit $X \sim \mathcal{U}([2, 5])$, la probabilité $\mathbb{P}(X \leq 4) = F_X(4)$ peut être obtenue avec la commande suivante :

```
punif(4, min = 2, max = 5)
[1] 0.6666667
```

Le fractile d'ordre α est retourné par la fonction `qunif`.

```
qunif(0.25, min = 2, max = 5) # fractile d'ordre 0.25, premier quartile
[1] 2.75
```

Enfin, la commande `runif` permet d'obtenir la réalisation d'une variable aléatoire suivant une loi uniforme.

```
runif(1, min = 2, max = 5)
[1] 3.67986
runif(1, min = 2, max = 5)
[1] 4.739771
runif(10, min = 2, max = 5) # permet d'obtenir la réalisation d'un
↪ échantillon de taille 10
[1] 2.348097 4.010598 2.547002 4.274435 4.589106 3.535062 3.260602
↪ 4.424660
[9] 3.020686 3.059558
```

⑥ À l'aide des fonctions introduites précédemment, calculer avec R les probabilités des événements suivants :

1. une variable normale centrée-réduite est supérieure à 3 ;
2. une variable normale d'espérance 35 et d'écart-type 6 est inférieure à 42 ;
3. une variable normale d'espérance 35 et d'écart-type 6 est comprise entre 40 et 50 ;
4. obtenir $n - 1$ faces sur n lancers d'une pièce de monnaie équilibrée, avec $n = 5, 10, 30$;
5. obtenir strictement plus de 14 faces sur 20 lancers d'une pièce de monnaie équilibrée ;
6. obtenir entre 10 et 15 faces sur 20 lancers d'une pièce de monnaie équilibrée.



```

# Question 1
1 - pnorm(3)
[1] 0.001349898
pnorm(3, lower.tail = FALSE)           # une autre manière d'obtenir le même résultat
[1] 0.001349898
# Question 2
pnorm(42, mean = 35, sd = 6)
[1] 0.8783275
pnorm((42 - 35)/6)                     # même résultat en passant par la loi normale centrée réduite
[1] 0.8783275
# Question 3
pnorm(50, mean = 35, sd = 6) - pnorm(40, mean = 35, sd = 6)
[1] 0.1961187
# Question 4
dbinom(4, 5, 0.5)
[1] 0.15625
dbinom(9, 10, 0.5)
[1] 0.009765625
dbinom(29, 30, 0.5)
[1] 2.793968e-08
# Question 5
1 - pbinom(14, 20, 0.5)
[1] 0.02069473
sum(dbinom(15:20, 20, 0.5))           # autre solution
[1] 0.02069473
# Question 6
pbinom(15, 20, 0.5) - pbinom(9, 20, 0.5)
[1] 0.5821896
sum(dbinom(10:15, 20, 0.5))          # deuxième solution
[1] 0.5821896

```

⑦ Calculer avec R les fractiles d'ordre $\alpha = 0.05, 0.1, 0.9$ pour les lois suivantes et les retrouver dans les tables statistiques du cours :



1. loi normale centrée réduite ;
2. loi du χ^2 à 10 degrés de liberté ;
3. loi de Student à 5 degrés de liberté ;
4. loi de Fisher à 2 et 5 degrés de liberté.

```

alpha <- c(0.05, 0.1, 0.9)
# Question 1
qnorm(alpha)
[1] -1.644854 -1.281552  1.281552
# Question 2
qchisq(alpha, 10)
[1]  3.940299  4.865182 15.987179
# Question 3
qt(alpha, 5)
[1] -2.015048 -1.475884  1.475884
# Question 4
qf(alpha, 2, 5)
[1] 0.05182311 0.10761220 3.77971608

```

3 Implémentation d'une loi de probabilité

On souhaite maintenant ajouter à R la loi de probabilité (densité, fonction de répartition, fonction quantile et simulation de v.a.) du premier exercice du TD n° 2 définie par la densité

de probabilité suivante :

$$f(x) = \begin{cases} ax & \text{si } 0 \leq x \leq b \\ 0 & \text{sinon} \end{cases},$$

avec $b > 0$ et $a = 2/b^2$. On notera cette loi $\mathcal{L}(b)$.

Sous R, il est possible de définir des fonctions avec la commande `function`. Par exemple, on peut définir la fonction `carre` qui à tout nombre x associe son carré x^2 de la manière suivante :

```
carre <- function(x) {
  y <- x * x
  return(y)
}
```

La fonction `carre` peut maintenant être utilisée comme une fonction de R.

```
carre(2)
[1] 4
carre(9)
[1] 81
```

Il est également possible de créer une fonction prenant plusieurs arguments. On peut construire la fonction `puissance` qui à deux nombres a et b associe a^b de la façon suivante :

```
puissance <- function(a, b) {
  return(a^b)
}
```

On obtient ainsi :

```
puissance(2, 3)
[1] 8
puissance(16, 1/2)
[1] 4
```

On peut remarquer qu'il est également possible de passer des vecteurs comme arguments des fonctions `carre` et `puissance`.

```
carre(c(1, 3, 5))
[1] 1 9 25
puissance(2:5, 3)
[1] 8 27 64 125
```

⑧ Écrire la fonction `dloi(x, b)` qui pour tout $x \in \mathbb{R}$ donne la valeur de la fonction de densité $f(x)$ de la loi $\mathcal{L}(b)$. On vérifiera que la fonction peut également prendre un vecteur en argument.

```
dloi <- function(x, b) {
  a <- 2/b^2
  f <- a * x
  f[x < 0] <- 0
  f[x > b] <- 0
  return(f)
}
```

La fonction `dloi` prend bien un vecteur en argument et donne la densité en chaque élément du vecteur. L'usage d'une boucle `for` et d'une structure de contrôle `if` est possible quoique inefficace et verbeuse.

- ⑨ On choisit $b = 3$, calculer la valeur de $f(x)$ pour $x = -1, 0, 1, 2, 3, 4, 5$.

```
dloi(-1:5, 3)
[1] 0.0000000 0.0000000 0.2222222 0.4444444 0.6666667 0.0000000 0.0000000
```

Il est possible de visualiser facilement la fonction `dloi` avec la commande suivante :

```
| curve(dloi(x, 3), from = -5, to = 5)
```

où les arguments `from` et `to` correspondent aux bornes entre lesquelles tracer la courbe.

- ⑩ Écrire la fonction `ploi(x, b)` qui donne la fonction de répartition F de la loi $\mathcal{L}(b)$.

La fonction de répartition est calculable directement à partir de la densité. On trouve

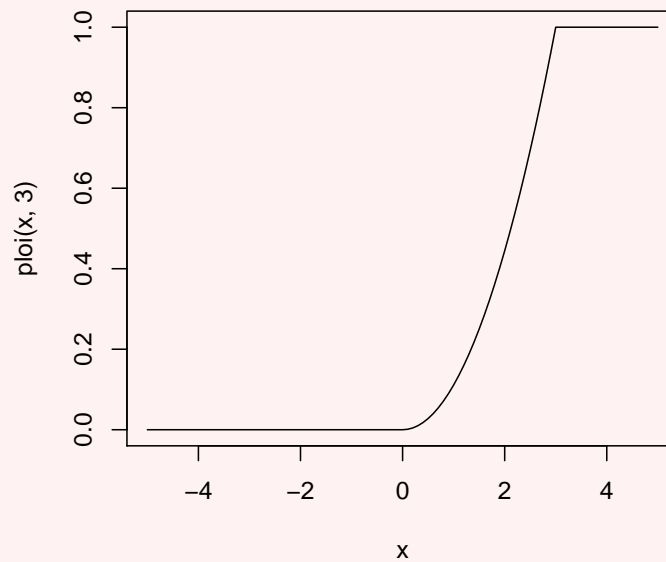
$$F(x) = \begin{cases} 0 & \text{si } x < 0 \\ \frac{x^2}{b^2} & \text{si } x \in [0; b] \\ 1 & \text{si } x > b, \end{cases}$$

d'où la fonction suivante.

```
ploi <- function(x, b) {
  F <- x^2/b^2
  F[x < 0] <- 0
  F[x >= b] <- 1
  return(F)
}
```

- ⑪ Tracer la fonction de répartition F entre -5 et 5 avec $b = 3$.

```
| curve(ploi(x, 3), from = -5, to = 5)
```



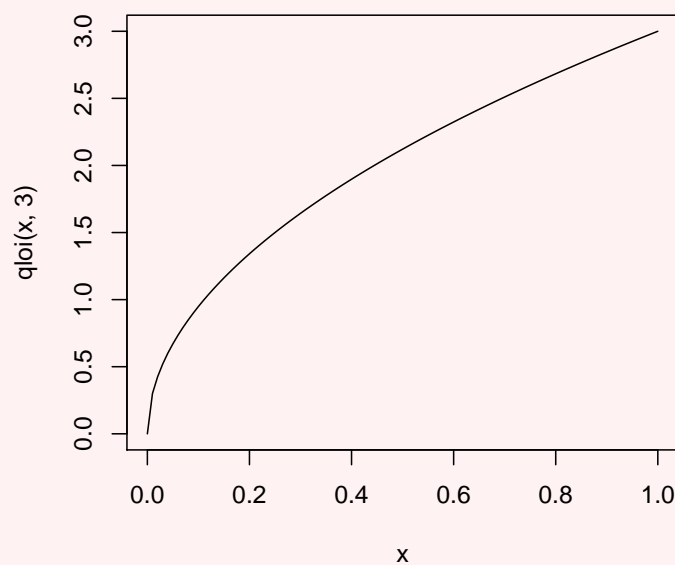
On observe bien la croissance de la fonction de répartition avec le morceau de parabole entre 0 et $b = 3$.

- 12) Écrire la fonction `qloi(alpha, b)` qui renvoie les fractiles $f_\alpha = F^{-1}(\alpha)$ de la loi $\mathcal{L}(b)$. On admettra que $F^{-1}(0) = 0$ et $F^{-1}(1) = b$.

On a la propriété suivante :

$$\forall \alpha \in]0; 1[, \quad F^{-1}(\alpha) = b\sqrt{\alpha}.$$

```
qloi <- function(alpha, b) {
  fa <- b * sqrt(alpha)
  fa[alpha == 0] <- 0
  fa[alpha == 1] <- b
  return(fa)
}
curve(qloi(x, 3), from = 0, to = 1)
```



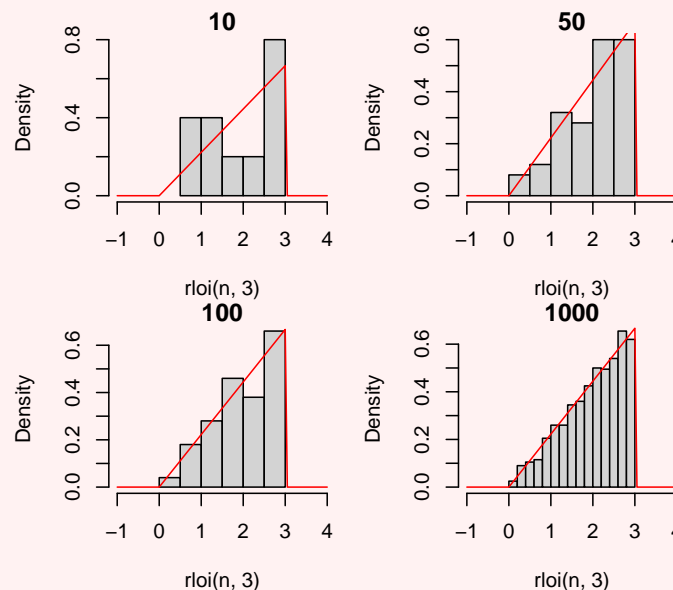
Soit $X \sim \mathcal{L}(b)$, on souhaite maintenant générer des réalisations de la variable aléatoire X .

- 13) Utiliser le résultat du quatrième exercice du TD n° 2 pour écrire la fonction `rloi(n, b)` qui génère aléatoirement un échantillon de taille n suivant la loi $\mathcal{L}(b)$.

```
rloi <- function(n, b) {
  u <- runif(n)
  x <- qlui(u, b)
  return(x)
}
```

- 14) Générer un échantillon de taille n suivant la loi $\mathcal{L}(b)$ et vérifier graphiquement que cette distribution empirique est proche de la loi $\mathcal{L}(b)$ quand n grandit. On prendra $b = 3$.

```
par(mfrow = c(2, 2))
for (n in c(10, 50, 100, 1000)) {
  hist(rloi(n, 3), breaks = round(1 + 10/3 * log10(n)), freq = FALSE, main = n, xlim = c(-1, 4))
  curve(dloi(x, 3), add = TRUE, col = "red")
}
```



La distribution empirique des échantillons semble coïncider avec la densité f à mesure que n grandit.

4 Bonus

On peut remarquer que la fonction `puissance` n'est définie pour les réels a et b . En effet, si $a < 0$, alors la fonction `puissance` n'est pas définie pour $b \in]-1; 1[$. De même, si $a = 0$, alors la fonction `puissance` n'est pas définie pour $b < 0$.

```
puissance(-1, 1/2)
[1] NaN
puissance(0, -2)
[1] Inf
```

La fonction `stop` permet d'afficher un message d'erreur et d'arrêter l'exécution d'une fonction. On peut compléter la fonction `puissance` ainsi :

```

puissance <- function(a, b) {
  if (any(a < 0) & b > -1 & b < 1)
    stop("non définie pour a < 0 et -1 < b < 1")
  if (any(a == 0) & b < 0)
    stop("non définie pour a = 0 et b < 0")
  return(a^b)
}

```

```

puissance(-1, 1/2)
Error in puissance(-1, 1/2): non définie pour a < 0 et -1 < b < 1
puissance(0:2, -2)
Error in puissance(0:2, -2): non définie pour a = 0 et b < 0

```

- 15) Corriger les fonctions dloi, ploi, qloi et rloi pour qu'elles affichent un message d'erreur lorsque leurs arguments sont mal définis.

```

dloi <- function(x, b) {
  if (b <= 0)
    stop("on doit avoir b > 0")
  a <- 2/b^2
  f <- a * x
  f[x < 0] <- 0
  f[x > b] <- 0
  return(f)
}
ploi <- function(x, b) {
  if (b <= 0)
    stop("on doit avoir b > 0")
  F <- x^2/b^2
  F[x < 0] <- 0
  F[x >= b] <- 1
  return(F)
}
qloi <- function(alpha, b) {
  if (b <= 0)
    stop("on doit avoir b > 0")
  if (any(alpha < 0) | any(alpha > 1))
    stop("on doit avoir 0 <= alpha <= 1")
  fa <- b * sqrt(alpha)
  fa[alpha == 0] <- 0
  fa[alpha == 1] <- b
  return(fa)
}
rloi <- function(n, b) {
  if (b <= 0)
    stop("on doit avoir b > 0")
  if (!is.integer(n) | n < 1)
    stop("n doit être un entier strictement positif")
  u <- runif(n)
  x <- qloi(u, b)
  return(x)
}

```