Fiche TD avec le logiciel R : tdr111

Comment simuler des tirages au hasard

J.R. Lobry

La fiche donne des indications pour utiliser un générateur de nombres pseudo-aléatoires avec une application au TT arbres (MathSV [BIO1004L]).

Table des matières

1 Introduction

Le logiciel R est un outil de pointe en statistiques, il s'utilise en première approximation comme une calculatrice. Entrez une commande, comme en rouge ci-dessous, au niveau du caractère d'invite de commande, `>`, dans la console de R :

```
 2 + 2
[1] 4
```

Ici, nous avons demandé à R de calculer `2 + 2` et il nous a répondu que cela valait `4`. Oubliez le `[1]` qui figure en début de la ligne la réponse : ce n'est qu'une commodité qui permet de repérer facilement le numéro des éléments quand la réponse est longue. Par exemple, si nous demandons la liste des lettres de l'alphabet :

```
 letters
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t"
[21] "u" "v" "w" "x" "y" "z"
```

Nous voyons donc ici que `"u"` est la 21^{e} lettre de l'alphabet, et sans avoir à recompter tout depuis le début nous constatons rapidement qu'il y a 26 lettres en tout dans l'objet `letters`. Vous n'avez pas besoin d'en savoir plus pour utiliser R, vous approfondirez l'utilisation de ce logiciel en deuxième année de licence.

Notez que vous pouvez directement copier les commandes en rouge dans ce document pour les coller dans la console R. Elles font l'objet d'un contrôle de qualité journalier, donc évitez de solliciter votre intervenant de TP pour dire que cela ne marche pas, toutes ces commandes ont été testées.

2 Simulation de jeux de hasard

2.1 Lancer d'un dé

À tout seigneur tout honneur, le mot hasard venant de l'arabe classique *yasara* pour jouer aux dés, nous commençons par la simulation du lancer d'un dé à six faces. Pour générer tous les entiers de 1 à 6 nous utilisons le caractère de ponctuation *deux points* (`:`) encadré par les valeurs de départ et d'arrivée :

```
 1:6
[1] 1 2 3 4 5 6
```

Nous rangeons cet ensemble de valeurs dans un objet appelé `dé` :

```
 1:6 -> dé
```

Pour examiner le contenu d'un objet il suffit d'entrer son nom dans la console :

```
 dé
[1] 1 2 3 4 5 6
```

Nous utilisons maintenant cet objet `dé` comme une urne dans laquelle nous faisons un tirage aléatoire, c'est-à-dire au hasard, grâce à la fonction `sample()`, *sample* signifiant échantillonner en anglais :

Un dé à jouer

```
sample(dé, 1)
```
```
[1] 6
```

Le `1` après la virgule dans `sample(dé, 1)` signifie simplement que l'on ne veut faire qu'un seul tirage. Pour faire plusieurs lancers de dé sans avoir à retaper la commande en entier il suffit d'utiliser la touche ↑ pour invoquer une instruction antérieure.

```
sample(dé, 1)
```
```
[1] 1
```
```
sample(dé, 1)
```
```
[1] 2
```
```
sample(dé, 1)
```
```
[1] 6
```

On peut également faire plusieurs tirages d'un coup :

```
sample(dé, 3)
```
```
[1] 3 6 2
```

2.2 Tirage à pile ou face

Lancer d'une pièce

Nous définissons un objet appelé `pièce` contenant les deux résultats possibles lors d'un lancer de pièce, puis nous l'utilisons comme une urne dans laquelle nous faisons un tirage :

```
c("pile", "face") -> pièce
pièce
```
```
[1] "pile" "face"
```
```
sample(pièce, 1)
```
```
[1] "pile"
```

On peut faire plusieurs tirages de suite mais il faut alors préciser que ces tirages se font avec remise, sans quoi notre urne sera rapidement vide puisqu'elle ne contient que deux éléments. Ceci se fait en ajoutant `replace = TRUE`, *replace = TRUE* signifiant remettre = VRAI en anglais :

```
sample(pièce, 20, replace = TRUE)
```
```
 [1] "face" "pile" "pile" "pile" "face" "face" "face" "pile" "face" "pile" "face"
[12] "face" "pile" "pile" "pile" "face" "face" "pile" "face" "pile"
```

2.3 Tirage d'un numéro à la roulette

Une roulette, dans sa version non-USA, comporte 37 cases numérotées de 0 à 36 :

```
0:36 -> roulette
roulette
```
```
 [1]  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
[28] 27 28 29 30 31 32 33 34 35 36
```

On fait 20 tirages avec remise et on range le résultat dans l'objet `RienNeVaPlus`, en référence à la phrase du croupier *Rien ne va plus* signifiant qu'il est trop tard pour faire une mise en anglais :

```
sample(roulette, 20, replace = TRUE) -> RienNeVaPlus
RienNeVaPlus
```
```
[1] 27  7  0 27 17 20 11 23 15  7 10  8 26 24 18 30  8 34  5 12
```

La fonction `table()` permet de compter le nombre de fois où chaque numéro a été obtenu :

```
table(RienNeVaPlus)
```
```
RienNeVaPlus
 0  5  7  8 10 11 12 15 17 18 20 23 24 26 27 30 34
 1  1  2  2  1  1  1  1  1  1  1  1  1  1  2  1  1
```

Une roulette

2.4 Tirage d'une carte au hasard

On définit une urne contenant les symboles d'un jeu de 32 cartes :

```
paste( c(7:10, "V", "D", "R", 1), rep(c("♠", "♡", "◇", "♣"), each =  8), sep = "") -> cartes
cartes
```
```
 [1] "7♠"  "8♠"  "9♠"  "10♠" "V♠"  "D♠"  "R♠"  "1♠"  "7♡"  "8♡"  "9♡"  "10♡" "V♡"
[14] "D♡"  "R♡"  "1♡"  "7◇"  "8◇"  "9◇"  "10◇" "V◇"  "D◇"  "R◇"  "1◇"  "7♣"  "8♣"
[27] "9♣"  "10♣" "V♣"  "D♣"  "R♣"  "1♣"
```

On tire une carte au hasard :

```
sample(cartes, 1)
```
```
[1] "1♠"
```

Comment simuler la distribution des cartes entre quatre joueurs ? Si on ne précise rien, la fonction `sample()` fait un tirage sans remise jusqu'à épuisement de l'urne, on range le résultat dans l'objet `donne` :

```
sample(cartes) -> donne
donne
```
```
 [1] "7◇"  "R♣"  "1♠"  "D♡"  "10◇" "9♡"  "7♣"  "V♡"  "1◇"  "1♡"  "9♠"  "R♡"  "8♡"
[14] "V♠"  "R♠"  "9♣"  "V◇"  "1♣"  "D◇"  "R◇"  "10♡" "9◇"  "7♠"  "D♣"  "10♠" "D♠"
[27] "7♡"  "10♣" "8♣"  "V♣"  "8◇"  "8♠"
```

Il suffit alors de répartir les cartes entre les quatre joueurs, par exemple en affichant le résultat comme un tableau de 4 colonnes :

```
matrix(donne, ncol = 4)
```
```
     [,1]  [,2] [,3]  [,4]
[1,] "7◇"  "1◇" "V◇"  "10♠"
[2,] "R♣"  "1♡" "1♣"  "D♠"
[3,] "1♠"  "9♠" "D◇"  "7♡"
[4,] "D♡"  "R♡" "R◇"  "10♣"
[5,] "10◇" "8♡" "10♡" "8♣"
[6,] "9♡"  "V♠" "9◇"  "V♣"
[7,] "7♣"  "R♠" "7♠"  "8◇"
[8,] "V♡"  "9♣" "D♣"  "8♠"
```

Un jeu de 32 cartes

3 Pour les curieux

Cette section d'aprofondissement peut être ignorée en première lecture : vous pouvez aller directement à la section 4 page 11 pour l'application au TT arbres.

3.1 est-il un bon simulateur ?

Une citation liminaire de VDM en guise de *captatio benevolentiae*.

> Aujourd'hui, tout fier, j'ai déclaré à ma compagne que j'ai réussi à faire ronronner notre chat. Elle m'a répondu d'un ton blasé : *T'emballe pas, il simule lui aussi.* [1]

Je pense avoir votre attention maintenant.

3.2 Déterminisme et reproductibilité

Nous venons de simuler quatre jeux de hasard avec un ordinateur, c'est-à-dire un système au comportement complètement déterministe. Il y a quelque chose d'un peu paradoxal ici. On peut se convaincre que les résultats donnés par R ne doivent rien au hasard en contrôlant la condition initiale du générateur de nombres aléatoires grâce à la fonction `set.seed()` :

```
set.seed(1)
matrix(sample(cartes), ncol = 4)
```

```
     [,1]  [,2]  [,3]  [,4]
[1,] "7♡"  "1♡"  "R♣"  "V♡"
[2,] "10♡" "8♠"  "R♡"  "9♢"
[3,] "8♢"  "V♠"  "10♣" "7♠"
[4,] "9♣"  "10♠" "9♡"  "8♣"
[5,] "D♠"  "D♡"  "1♢"  "V♢"
[6,] "7♣"  "1♠"  "9♠"  "D♢"
[7,] "V♣"  "10♢" "R♠"  "D♣"
[8,] "7♢"  "1♣"  "R♢"  "8♡"
```

```
set.seed(1)
matrix(sample(cartes), ncol = 4)
```

```
     [,1]  [,2]  [,3]  [,4]
[1,] "7♡"  "1♡"  "R♣"  "V♡"
[2,] "10♡" "8♠"  "R♡"  "9♢"
[3,] "8♢"  "V♠"  "10♣" "7♠"
[4,] "9♣"  "10♠" "9♡"  "8♣"
[5,] "D♠"  "D♡"  "1♢"  "V♢"
[6,] "7♣"  "1♠"  "9♠"  "D♢"
[7,] "V♣"  "10♢" "R♠"  "D♣"
[8,] "7♢"  "1♣"  "R♢"  "8♡"
```

On constate que l'on a obtenu exactement le même tirage. C'est pourquoi on parle de tirage pseudo-aléatoires. Si on change la condition initiale on obtient des tirages différents :

```
set.seed(2)
matrix(sample(cartes), ncol = 4)
```

```
     [,1]  [,2]  [,3]  [,4]
[1,] "D♠"  "10♡" "10♢" "9♠"
[2,] "D♢"  "V♡"  "9♢"  "7♢"
[3,] "8♢"  "R♢"  "R♠"  "R♣"
[4,] "V♠"  "1♣"  "7♠"  "8♡"
[5,] "9♣"  "1♡"  "D♣"  "D♡"
[6,] "8♣"  "10♣" "V♣"  "7♣"
[7,] "10♠" "1♠"  "7♡"  "1♢"
[8,] "V♢"  "R♡"  "8♠"  "9♡"
```

```
set.seed(2)
matrix(sample(cartes), ncol = 4)
```

```
     [,1]  [,2]  [,3]  [,4]
[1,] "D♠"  "10♡" "10♢" "9♠"
[2,] "D♢"  "V♡"  "9♢"  "7♢"
[3,] "8♢"  "R♢"  "R♠"  "R♣"
[4,] "V♠"  "1♣"  "7♠"  "8♡"
[5,] "9♣"  "1♡"  "D♣"  "D♡"
[6,] "8♣"  "10♣" "V♣"  "7♣"
[7,] "10♠" "1♠"  "7♡"  "1♢"
[8,] "V♢"  "R♡"  "8♠"  "9♡"
```

[1] Source : `http://www.viedemerde.fr/inclassable/8453844`

3.3 Propriétés souhaitables d'un tirage pseudo-aléatoire

Ce que l'on souhaite c'est que les tirages pseudo-aléatoires se comportent comme si on avait un tirage réellement aléatoire. Reprenons le cas du lancer de dé. Il est raisonnable de souhaiter que le dé ne soit pas truqué et que toutes les faces aient la même probabilité, $\frac{1}{6}$, d'être obtenues. Est-ce le cas ? Faisons une expérience, reproductible en invoquant `set.seed(1)` au préalable, en lançant le dé six-cent-mille (600000) fois :

```
set.seed(1)
table(sample(dé, 600000, replace = TRUE)) -> obs
obs
```

```
     1      2      3      4      5      6
100129  99924  99734 100389 100207  99617
```

Dans cette expérience nous avons donc obtenu :

La face 1 cent-mille-cent-vingt-neuf (100129) fois.

La face 2 quatre-vingt-dix-neuf-mille-neuf-cent-vingt-quatre (99924) fois.

La face 3 quatre-vingt-dix-neuf-mille-sept-cent-trente-quatre (99734) fois.

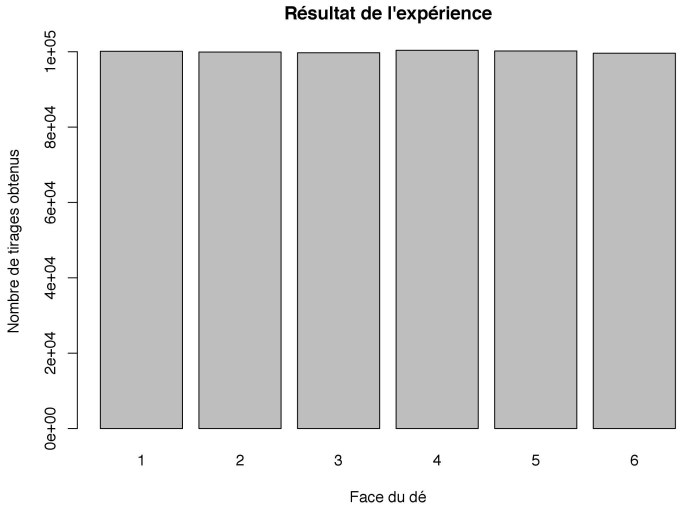
La face 4 cent-mille-trois-cent-quatre-vingt-neuf (100389) fois.

La face 5 cent-mille-deux-cent-sept (100207) fois.

La face 6 quatre-vingt-dix-neuf-mille-six-cent-dix-sept (99617) fois.

Une représentation graphique valant généralement mieux qu'un long discours on peut également résumer les résultats de notre expérience avec un diagramme en bâtons :

```
barplot(obs, main = "Résultat de l'expérience", xlab = "Face du dé",
        ylab ="Nombre de tirages obtenus")
```



Nous pouvons donc constater que le dé n'est pas outrageusement truqué puisque nous avons obtenu à peu près chaque face cent-mille (100000) fois, soit avec une probabilité voisine de $\frac{1}{6}$, puisque nous avons fait six-cent-mille (600000) lancers en tout.

3.4 Le dé est-il subtilement truqué ?

Vous avez vu en TD que l'on pouvait faire mieux que cette simple assertion comme quoi le dé n'est pas grossièrement truqué. Il s'agit de comparer la distribution observée ici avec celle théoriquement attendue quand le dé n'est pas pipé, soit chaque face cent-mille (100000) fois. Nous allons donc faire un test d'hypothèse, avec comme hypothèse nulle, H_0 : *le dé n'est pas truqué*, contre l'hypothèse alternative, H_1 : *le dé est truqué*. Nous choisissons classiquement de fixer le risque de première espèce, le rejet à tort de l'hypothèse nulle, à 5 %, soit $\alpha = 0.05$.

La mise en œuvre du test consiste à calculer un indice d'écart entre les deux distributions,

$$\chi^2_{\text{obs}} = \sum_{i=1}^{6} \frac{(\text{obs}_i - \text{theo}_i)^2}{\text{theo}_i},$$

où obs_i représente le nombre de fois où la face i a été observée, et theo_i le nombre de fois où la face i était attendue sous l'hypothèse nulle d'un dé non truqué, soit dans notre cas cent-mille (100000) fois, et ce quelle que soit la face. Le calcul de l'écart χ^2_{obs} se fait très directement sous R de la façon suivante :

```
100000 -> theo
sum((obs - theo)^2/theo) -> chi2obs
chi2obs
```

```
[1] 4.34032
```

Nous obtenons donc $\chi^2_{\text{obs}} = 4.34$ comme valeur de l'indice d'écart entre les deux distributions. La question est de savoir si la valeur de cet indice est anormalement élevée ou pas. La valeur seuil lue dans la table du χ^2 est de 11.070 (*cf* figure 1 page 8). Graphiquement, la signification de cette valeur est donnée dans la figure 2 page 9. La valeur observée n'est donc pas anormalement élevée, nous sommes donc dans l'incapacité de rejeter l'hypothèse nulle. Nous décidons que le dé n'est pas truqué avec un risque de deuxième espèce, β, inconnu.

Vous vous doutez bien que lorsque l'on travaille avec un logiciel de pointe comme R on n'utilise plus de tables statistiques antédiluviennes (plus personne n'utilise de tables statistiques dans la vraie vie, sauf à des fins pédagogiques). Le test d'ajustement à une distribution uniforme se fait tout simplement ainsi :

```
chisq.test(obs) -> résultat
```

Les résultats du test sont enregistrés ici dans l'objet `résultat`. C'est un objet qui comporte de nombreux éléments désignés par leur nom :

```
names(résultat)
```

```
[1] "statistic" "parameter" "p.value"   "method"    "data.name" "observed"
[7] "expected"  "residuals" "stdres"
```

On peut accéder aux éléments qui nous intéressent en utilisant l'opérateur `$`, par exemple `observed` donne les effectifs observés, c'est-à-dire le résultat de notre expérience de lancer de dé :

```
résultat$observed
```

```
     1      2      3      4      5      6
100129  99924  99734 100389 100207  99617
```

Table de χ^2.

La table donne, pour une probabilité α et un nombre de degrés de liberté (d.d.l.) donnés, la valeur x telle que la probabilité que χ^2 égale ou dépasse x vaut α.

d.d.l. \ α	0.90	0.50	0.30	0.20	0.10	0.05	0.02	0.01	0.001
1	0.016	0.455	1.074	1.642	2.706	3.841	5.412	6.635	10.828
2	0.211	1.386	2.408	3.219	4.605	5.991	7.824	9.210	13.816
3	0.584	2.366	3.665	4.642	6.251	7.815	9.837	11.345	16.266
4	1.064	3.357	4.878	5.989	7.779	9.488	11.668	13.277	18.467
5	1.610	4.351	6.064	7.289	9.236	11.070	13.388	15.086	20.515
6	2.204	5.348	7.231	8.558	10.645	12.592	15.033	16.812	22.458
7	2.833	6.346	8.383	9.803	12.017	14.067	16.622	18.475	24.322
8	3.490	7.344	9.524	11.030	13.362	15.507	18.168	20.090	26.124
9	4.168	8.343	10.656	12.242	14.684	16.919	19.679	21.666	27.877
10	4.865	9.342	11.781	13.442	15.987	18.307	21.161	23.209	29.588

FIGURE 1 – **Lecture de la valeur seuil 11.070**. Extrait de l'annexe de votre polycopié de TD. Nous avons choisi de fixer le risque de première espèce à 5 %, soit $\alpha = 0.05$, nous retenons donc la colonne surlignée en jaune. Le nombre de degrés de liberté (d.d.l.) est égal au nombre de classes après regroupement moins un, moins le nombre de paramètres estimés, soit ici pour notre expérience avec un dé à six faces, 6 - 1 - 0 = 5, aussi nous retenons la ligne surlignée en jaune. À l'intersection nous lisons la valeur seuil de 11.070.

Les effectifs théoriques sont dans `expected`, on retrouve ici la valeur de centmille (10^5 noté `1e+05` dans la sortie de R) attendue pour chaque face du dé s'il n'est pas truqué :

```
résultat$expected
```

```
    1     2     3     4     5     6
1e+05 1e+05 1e+05 1e+05 1e+05 1e+05
```

On retrouve dans `statistic` notre indice d'écart, χ^2_{obs}, entre les deux distributions :

```
résultat$statistic
```

```
X-squared
  4.34032
```

Plutôt que de comparer cette valeur avec une valeur seuil lue dans une table du χ^2 comme précédemment on exploite la valeur donnée par `p.value` :

```
résultat$p.value
```

```
[1] 0.5015279
```

La signification graphique de cette valeur p est donnée dans la figure 3 page 10. Le premier avantage de procéder ainsi est que plutôt que de lire une valeur dans une table il suffit pour prendre une décision de comparer directement cette valeur p avec le risque de première espèce, α, choisi. Si la valeur p est inférieure à α on rejette H_0, sinon on ne la rejette pas. Comme nous avons choisi $\alpha = 0.05$ nous ne rejetons pas H_0, toujours avec un risque de deuxième

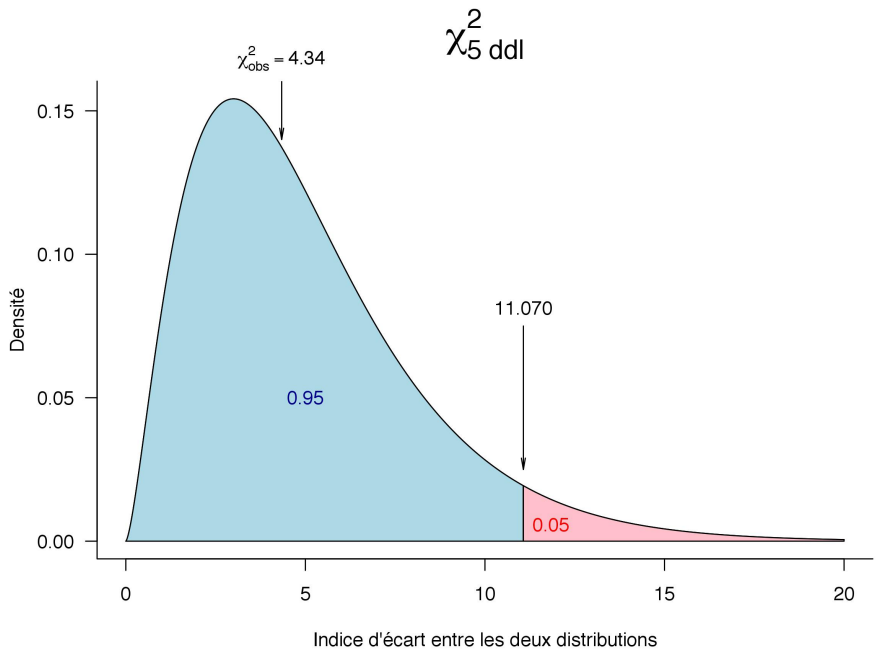


FIGURE 2 – **Signification de la valeur seuil 11.070**. La courbe donne la fonction de densité de probabilité de la loi du $\chi^2_{5\text{ddl}}$. Sous H_0, c'est-à-dire si le dé n'est pas truqué, notre indice d'écart est distribué selon cette loi du χ^2 à 5 degrés de liberté. La valeur seuil de 11.070 est telle que la distribution est coupée en deux avec à gauche en bleu 95 % des valeurs et à droite en rouge les 5 % restant. Cela correspond bien à un risque de première espèce $\alpha = 0.05$ puisque qu'avec cette valeur seuil nous allons rejeter H_0 à tort dans 5 % des cas.

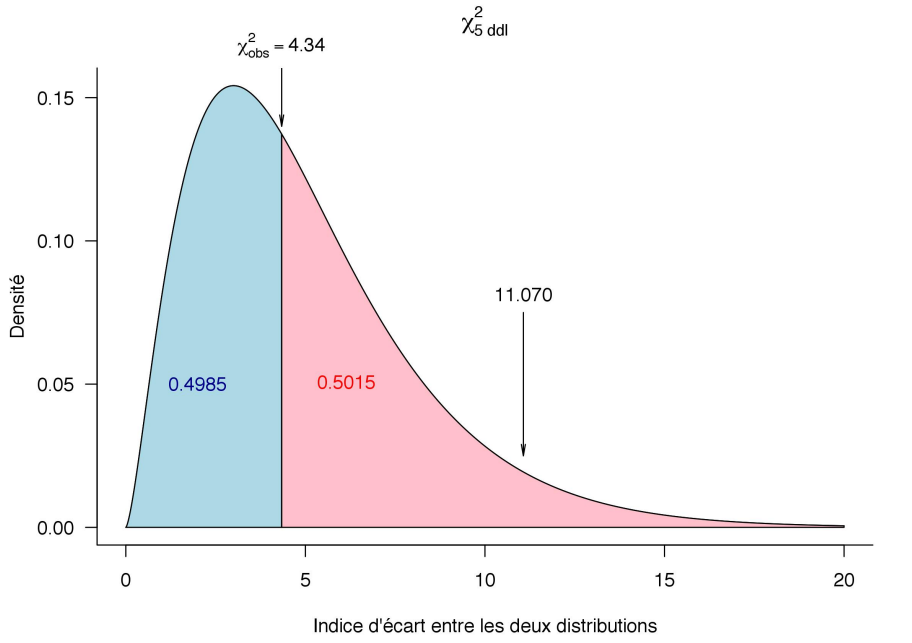


FIGURE 3 – **Signification de la valeur $p = 0.5015$**. La courbe donne la fonction de densité de probabilité de la loi du $\chi^2_{5\text{ddl}}$. Sous H_0, c'est-à-dire si le dé n'est pas truqué, notre indice d'écart est distribué selon cette loi du χ^2 à 5 degrés de liberté. La valeur p nous donne directement comment la distribution est coupée en deux : on s'attend sous H_0 à avoir, à droite en rouge, 50.15 % des valeurs plus grandes que notre valeur observée et par conséquent à gauche en bleu 49.85 % des valeurs inférieures

espèce, β, inconnu. Notez que cela revient exactement au même que précédemment puisque le changement de décision se fera quand le χ^2_{obs} dépassera la valeur seuil de 11.070 puisque que c'est à partir de là que la valeur p sera inférieure à 0.05.

Le deuxième avantage est que nous ne sommes pas limités pour le choix du risque de première espèce aux neuf valeurs pré-définies dans la table (*cf* figure 1 page 8). Nous pourrions vouloir pour suivre des préconisations récentes [1] choisir un risque de première espèce $\alpha = 0.005$. Avec la table, c'est impossible, alors qu'avec la valeur p c'est immédiat. Tout se passe comme si nous avions à notre disposition une table du χ^2 avec un nombre infini de colonnes.

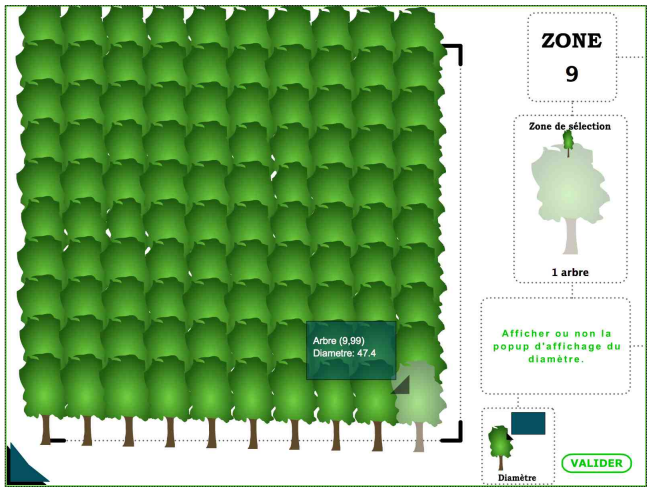
3.5 est un bon simulateur

L'équiprobabilité des tirages n'est pas la seule propriété souhaitable pour un générateur de nombres pseudo-aléatoires. Dans le cas d'un tirage à pile ou face si on obtient alternativement des piles et des faces on aura bien équiprobabilité mais c'est trop prévisible d'un tirage à l'autre pour être considéré comme une bonne simulation du hasard. Sans entrer dans les détails, on peut résumer en

disant que le générateur de nombres pseudo-aléatoires de R a de très bonnes propriétés parce qu'il a été développé par des spécialistes du domaine [2].

4 Application au TT arbres

Chaque forêt compte 1000 arbres répartis en 10 zones numérotées de 0 à 9. Chaque zone comporte 100 arbres numérotés de 0 à 99. Le dernier arbre est donc noté (9, 99) :



On peut faire un tirage aléatoire simple en tirant au hasard, sans remise, un nombre entre 000 et 999 avec, pour convention, que le premier chiffre donne le numéro de la zone et les deux suivants le numéro de l'arbre. Supposons que dans un premier temps nous voulions échantillonner 30 arbres :

```
set.seed(1)
sample(0:999, 30)
```

```
 [1] 265 371 571 905 200 893 939 656 624  61 203 174 678 379 759 490 706 975 373 762
[21] 916 207 637 122 260 376  13 372 845 330
```

Pour faciliter la lecture on divise toutes les valeurs par cent afin de mettre en évidence le numéro de la zone et celui de l'arbre :

```
set.seed(1)
sample(0:999, 30)/100
```

```
 [1] 2.65 3.71 5.71 9.05 2.00 8.93 9.39 6.56 6.24 0.61 2.03 1.74 6.78 3.79 7.59 4.90
[17] 7.06 9.75 3.73 7.62 9.16 2.07 6.37 1.22 2.60 3.76 0.13 3.72 8.45 3.30
```

Pour faciliter le travail on trie les valeurs par ordre croissant avec la fonction `sort()`, *sort* signifiant trier en anglais, afin de regrouper les arbres par zone :

```
set.seed(1)
sort(sample(0:999, 30)/100)
```

```
 [1] 0.13 0.61 1.22 1.74 2.00 2.03 2.07 2.60 2.65 3.30 3.71 3.72 3.73 3.76 3.79 4.90
[17] 5.71 6.24 6.37 6.56 6.78 7.06 7.59 7.62 8.45 8.93 9.05 9.16 9.39 9.75
```

Vous pouvez maintenant exploiter ce tirage aléatoire simple qui présente en plus l'avantage d'être reproductible puisque nous avons utilisé `set.seed()` avant le tirage.

Pour compter le nombre d'arbres par zone on commence par extraire les numéros de zone avec la fonction partie entière `floor()`, *floor* signifiant plancher en anglais, pour l'entier juste en dessous d'un nombre réel :

```
set.seed(1)
floor(sort(sample(0:999, 30)/100))
```

```
[1] 0 0 1 1 2 2 2 2 2 3 3 3 3 3 4 5 6 6 6 6 7 7 7 8 8 9 9 9 9
```

La fonction `table()` compte alors le nombre d'arbres par zone échantillonée, notez qu'avec un tirage aléatoire simple il n'y a pas forcément le même nombre d'arbres par zone :

```
set.seed(1)
table(floor(sort(sample(0:999, 30)/100)))
```

```
0 1 2 3 4 5 6 7 8 9
2 2 5 6 1 1 4 3 2 4
```

L'avantage d'un tirage reproductible est sensible si vous avez besoin de compléter votre série. Supposons par exemple que vous décidez d'échantillonner 10 arbres en plus. Il ne faut pas choisir ceux qui étaient déjà dans votre premier échantillon de 30 arbres. Rien n'est plus simple que d'allonger le tirage de 10 éléments :

```
set.seed(1)
sample(0:999, 40)/100
```

```
 [1] 2.65 3.71 5.71 9.05 2.00 8.93 9.39 6.56 6.24 0.61 2.03 1.74 6.78 3.79 7.59 4.90
[17] 7.06 9.75 3.73 7.62 9.16 2.07 6.37 1.22 2.60 3.76 0.13 3.72 8.45 3.30 4.67 5.80
[33] 4.77 1.80 7.99 6.45 7.65 1.03 6.96 3.95
```

Les trente premiers ne nous intéressent pas puisque nous les avions déjà. On peut utiliser les opérateurs d'indexation de R pour extraire les éléments de 31 à 40 :

```
set.seed(1)
(sample(0:999, 40)/100)[31:40]
```

```
[1] 4.67 5.80 4.77 1.80 7.99 6.45 7.65 1.03 6.96 3.95
```

Il ne reste plus qu'à trier pour se faciliter la tâche :

```
set.seed(123)
sort((sample(0:999, 40)/100)[31:40])
```

```
[1] 0.23 2.08 2.22 3.06 4.61 6.68 7.31 7.69 8.74 9.34
```

Voici donc 10 arbres tirés au hasard complétant notre échantillon précédent. La seule chose à bien noter c'est la valeur utilisée dans `set.seed()` pour pouvoir reproduire les résultats ultérieurement.

Références

[1] V.E. Johnson. Revised standards for statistical evidence. *Proceedings of the National Academy of Sciences of the United States of America*, 110 :19313–19317, 2013.

[2] Matsumoto M. and Nishimura T. Mersenne Twister : A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8 :3–30, 1998.