Correction TD Simulation variables aléatoires

Travaux dirigés

Pierre Gloaguen

Contents

Packages

Vous utiliserez la suite de paquet `tidyverse`

```
library(tidyverse) # Ensemble de packages pour la manip de données
```

Sur certains mac, selon ne fonctionne pas, vpous chargerez alors les paquets suivants (qui sont inclus dans tidyverse).

```
library(dplyr) # Pour la manipulation des tableaux
library(tidyr) # Pour les fonctions spread et gather
library(purrr) # Pour les fonctions map (vectorisation)
library(ggplot2) # Pour les graphes
```

1 Générateurs pseudos aléatoires

1.1 Génération de loi uniforme

1. À l'aide du logiciel R, programmez un générateur à congruences pour la loi uniforme. Ce générateur prendra la forme d'une fonction prenant en argument:

- Un entier `n` donnant la taille de l'échantillon voulu.
- 4 entiers `a, m, c, x0` correspondant aux paramètres du générateurs vu en cours.

```
rm(list = ls())# Nettoyage de l'environnement de travail
mon_runif <- function(n , a, m, c, x0){
  # %% est l'opérateur modulo
  x0 <- x0 %% m# On s'assure que x0 est bien plus petit que m
  xs <- rep(x0, n + 1)# Suite des xs, on ne gardera que les n derniers
  for(k in 2:(n+1)){
    xs[k] <- (a * xs[k - 1] + c) %% m
  }
  us <- xs / m # Mise entre 0 et 1
  # Retour sous forme de data.frame, pratique pour les ggplot
  return(tibble(n = 1:n, u = us[-1]))# On ne retourne pas x0 / m
}
```

```
ech1 <- mon_runif(n = 10000, a = 41358, m =  2^31 - 1,
                  c = 0, x0 = 1)
```

2. À l'aide de cette fonction, générer un échantillon de taille 10000 pour les valeurs

- a = 41358
- m = 2^31 -1
- c = 0

et la graine de votre choix. Refaites la même procédure avec

- a = 3
- m = 2^31 -1
- c = 0

et

- a = 101
- m = 2311
- c = 0

Vous stockerez chacun des échantillons obtenus.

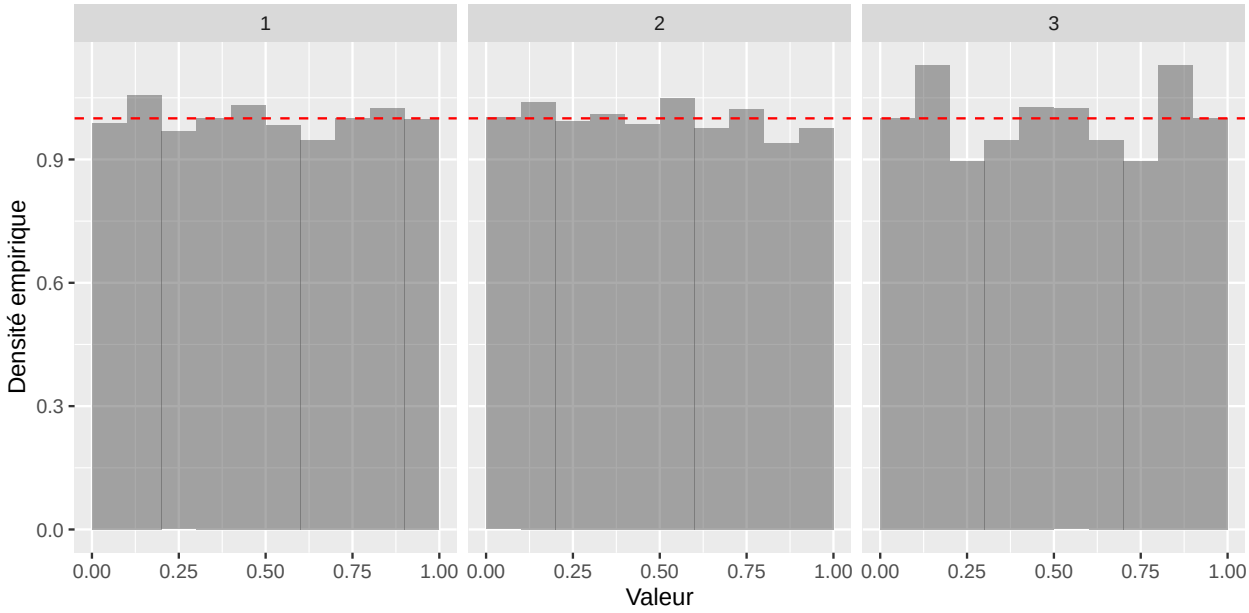
```
parametres <- list(a = list(41358, 3, 101),# On stocke les différents paramètres
                   m = list(2^31 - 1, 2^31 - 1, 2311))# dans une lise
# On peut faire plusieurs traitements sans boucle dans R
resultats <- purrr::pmap_dfr(parametres, # On applique à la liste des paramètres
                             mon_runif, # La fonction
                             n = 10000, c = 0, x0 = 17, # Les paramètres manquants
                             .id = "generateur")# On garde le generateur en memoire
resultats
```

```
# A tibble: 30,000 x 3
   generateur     n        u
   <chr>      <int>    <dbl>
 1 1              1 0.000327
```

```
 2 1              2 0.541
 3 1              3 0.399
 4 1              4 0.0787
 5 1              5 0.789
 6 1              6 0.704
 7 1              7 0.905
 8 1              8 0.805
 9 1              9 0.282
10 1             10 0.0132
# ... with 29,990 more rows
```

3. Pour chacun des échantillons obtenus, tracez l'histogramme empirique. Quels échantillons vous semblent tirés selon une loi uniforme $U[0,1]$? En utilisant la fonction `ks.test`, effectuez un test de Kolmogorov-Smirnoff d'adéquation pour la loi uniforme. Que concluez vous sur la qualité des 3 générateurs?

```
ggplot(resultats) + # Tableau à représenter
  aes(x = u) + # Abscisse commune
  geom_histogram(mapping = aes(y = ..density..),# L'aire des rectangles somme à 1
                 breaks = seq(0, 1, by = 0.1), # Breaks de l'histogramme
                 alpha = 0.5) + # Transparence
  geom_hline(yintercept = 1, linetype = 2, col = "red") + # Ajout de y = 1
  labs(x = "Valeur", y = "Densité empirique") + # Habillage
  facet_grid(. ~ generateur)# Un graphe par generateur
```



\begin{Correction}

Les deux échantillons ont l'air de couvrir à peu près uniformément l'intervalle [0, 1]. Le 3ème par contre semble poser problème. On peut regarder le résultats du test d'adéquation de Kolmogorov Smirnoff. On ne regarde ici que les probabilités critiques du test (effectués par numéro, d'où le \texttt{group_by} avec la fonction `ks.test`).

\end{Correction}

```
kolmog_smirnoff_test_p_values <- resultats %>%
  group_by(generateur) %>% # On regroupe les genrateurs ensembles
  # pour chaque generateur, on effectue le test de kolmogoroff smirnoff
```

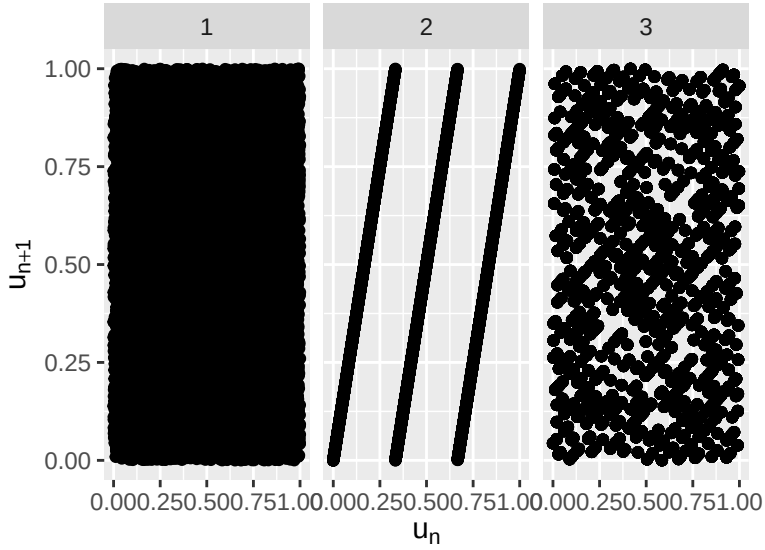
```r
  # on extrait la p valeur et on la stocke dans p_val
  summarise(p_val = ks.test(u, y = "punif")$p.value)
kolmog_smirnoff_test_p_values
```

```
# A tibble: 3 x 2
  generateur   p_val
  <chr>        <dbl>
1 1          0.757
2 2          0.294
3 3          0.00251
```

Au risque de première espèce $\alpha = 1\%$, on rejette H_0 pour le troisième générateur.

4. Pour chacun des échantillons $(u_1, \ldots, u_{10000})$ obtenus, tracez u_n en fonction de u_{n-1}. Que pouvez vous conclure sur la qualité des 3 générateurs?

```r
resultats %>%
  group_by(generateur) %>%
  # Pour chaque generateur, on cree une colonne u_n_plus1  comprenant u[n+1]
  mutate(u_n_plus1 = lead(u)) %>% # mutate est une fonction de creation de colonne
  ggplot() + # on représente graphiquement le resultat
  aes(x = u, y = u_n_plus1) + # u[n+1] en fonction de u[n]
  geom_point() + # On représente un point par couple
  facet_wrap(.~generateur) + # Un graphe par generateur
  labs(x = expression(u[n]), y = expression(u[n + 1])) # Habillage
```



2 Méthode d'inversion

2.1 Loi exponentielle

On rappelle qu'une variable aléatoire X est de loi exponentielle de paramètre $\lambda > 0$ si elle a pour fonction de densité $f_X(x) = \lambda e^{-\lambda x} \mathbf{1}_{x \geq 0}$

1. En utilisant la méthode d'inversion, proposez un algorithme de simulation pour une variable aléatoire exponentielle.

On a que

$$\begin{aligned} F(x) &= \int_{-\infty}^{x} \lambda \mathrm{e}^{-\lambda z} \mathbf{1}_{z \geq 0} dz \\ &= 1 - \mathrm{e}^{-\lambda x} \end{aligned}$$

Cette fonction est continue et bijective de $\mathbb{R}^+$ dans $]0,1[$, son inverse est, pour $u \in]0,1[$

$$F^{-1}(u) = -\frac{\ln(1-u)}{\lambda}$$

Ainsi, si on simule une variable aléatoire $U \sim \mathcal{U}[0,1]$, alors $X = -\frac{\ln(1-U)}{\lambda}$ est distribuée selon une variable aléatoire exponentielle de paramètre λ.

2. Ecrire une fonction `R` mettant en oeuvre cette algorithme. Cette fonction prendra deux paramètres en entrée:

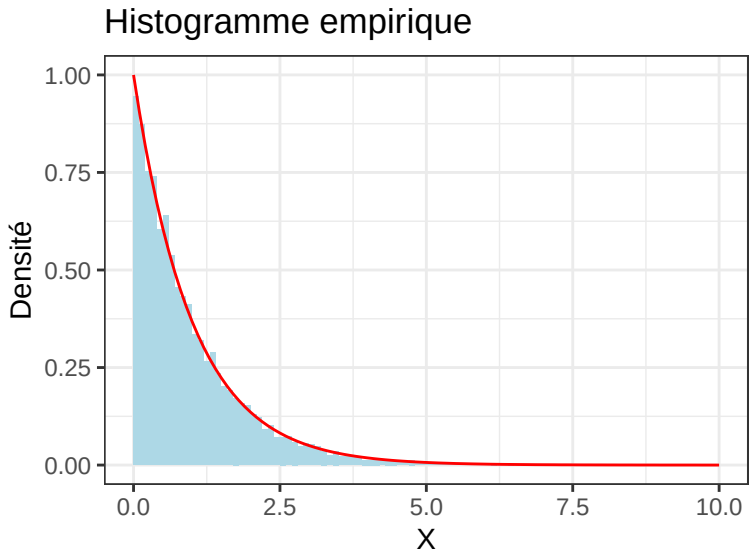
- `n` La taille de l'échantillon;
- `lambda` Le paramètre de la loi exponentielle

Vous testerez la qualité de votre fonction sur un échantillon de taille 10000, en comparant graphiquement l'histogramme empirique obtenu à la densité de la loi exponentielle correspondante.

```
mon_rexp <- function(n, lambda){
  us <- runif(n)# Vecteur d'uniformes
  echantillon = - log(1 - us) / lambda
  return(tibble(n = 1:n, valeur = echantillon))
}
```

On simule un échantillon, et on en trace le graphe

```
set.seed(123)# Fixe la graine aléatoire dans runif, pour reproductibilité
lambda <- 1
echantillon_exp <- mon_rexp(1e4, lambda)
my_breaks <- seq(0, 10, by = 0.1)# Point de ruptures de l'histogramme
ggplot(echantillon_exp,  mapping = aes(x = valeur)) +
  geom_histogram(mapping = aes(y = ..density..), fill = "lightblue",
                 breaks = my_breaks) +
  geom_line(data = data.frame(x = my_breaks, y = dexp(my_breaks, lambda)),
            mapping = aes(x = x, y= y), col = "red") +
  theme_bw() + labs(x = "X", y = "Densité", title = "Histogramme empirique")
```



2.2 Loi discrète

On considère une variable aléatoire discrète X à valeurs dans l'ensemble $\{1, \dots, K\}$, dont la loi est définie par le vecteur de probabilité $(p_1, \dots, p_K)$, i.e.:

$$\mathbb{P}(X = k) = p_k \tag{1}$$

$$\sum_{k=1}^{K} p_k = 1 \tag{2}$$

1. Pour tout $u \in]0, 1[$, écrire l'expression de l'inverse généralisée de la fonction de répartition de X.

$$F(x) = \sum_{k=1}^{K} p_k \mathbf{1}_{k \leq x}$$

$$F^{-1}(u) = \begin{cases} 1 & si \quad 0 < u \leq p_1 \\ 2 & si \quad p_1 < u \leq p_1 + p_2 \\ \dots & \quad \dots \\ k & si \quad \sum_{i=1}^{k-1} p_i < u \leq \sum_{i=1}^{k} p_i \\ \dots & \quad \dots \\ K & si \quad \sum_{i=1}^{K-1} p_i < u < 1 \end{cases}$$

2. En déduire un algorithme de simulation pour toute variable aléatoire discrète dans un ensemble fini.

Ainsi, il suffit de tirer $U \sim \mathcal{U}[0, 1]$ et de trouver l'indice k tel que $\sum_{i=1}^{k-1} p_i < u \leq \sum_{i=1}^{k} p_i$

3. Utilisez cet algorithme de simulation pour simuler un échantillon de taille 10000 loi binomiale de paramètres $n = 10$ et $p = 0.5$ avec `R`. Vous comparerez les fréquences obtenues avec les fréquences théoriques.

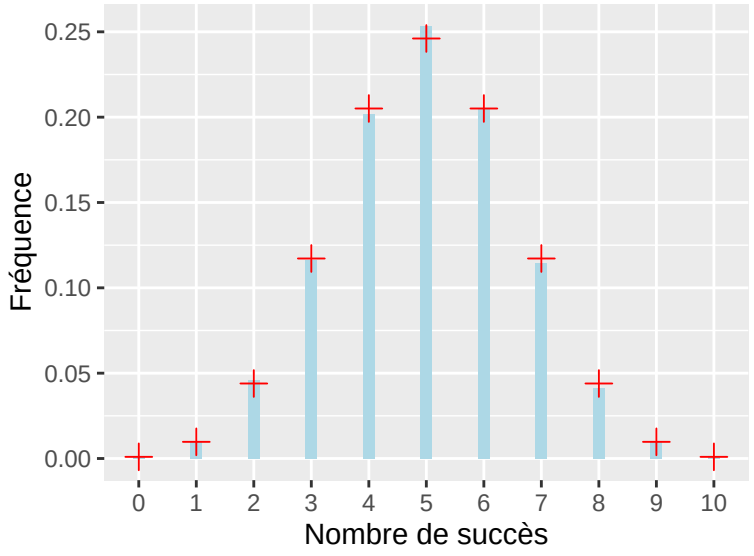
On utilisera `dbinom` pour calculer les probabilités d'une loi binomiale. Ensuite, la fonction findInterval évite de faire une boucle.

```r
mon_rbinom <- function(n, n_binom, p){
  # Pour les probas on utilise la fonction dbinom
  (vecteur_probs <- c(0, dbinom(0:n_binom, size = n_binom, prob = p)))
  (prob_cumule <- cumsum(vecteur_probs)) # Probas cumules)
  us <- runif(n)
  # La fonction findInterval permet de trouver dans quel intervalle se trouve
  # un u
  intervalles <- findInterval(us, prob_cumule, rightmost.closed = T,
                              left.open = T)
  # Il faut retrancher 1 à "intervalles" pour obtenir les valeurs attendues
  # d'une binomiale
  return(data.frame(n = 1:n,
                    valeur = factor(intervalles - 1, levels = 0:n_binom)))
}
```

```r
# Représentation graphique
set.seed(123) # Pour la reproductibilité
n_sample <- 1e4; n_binom <- 10; p <- 0.5 # Parametre
echantillon_binom <- mon_rbinom(n_sample, n_binom, p) # Echantillon génére
```

```r
# Calcul et stockage des fréquences empiriques
frequences_empiriques <- echantillon_binom %>%
  group_by(valeur, .drop = FALSE) %>% # On regroupe par valeur de nbre de succes
  summarise(freq_empir = n() / n_sample) %>% # On divise le nombre total n() par n_sample
  ungroup() %>% # On dégroupe
  mutate(freq_theo = dbinom(0:n_binom, n_binom, p)) # On ajoute la fréquence theorique
```

```r
# Représentation graphique
ggplot(frequences_empiriques) + # Tableau à représenter
  aes(x = valeur) + # Abscisse commune
  geom_col(mapping = aes(y = freq_empir), # Ordonnée des fréquences empiriques
           width = 0.2, fill = "lightblue") +
  geom_point(aes(y = freq_theo), # On ajoute le point des fréquences théoriques
             shape = 3, col = "red", size = 3) +
  labs(y = "Fréquence", x = "Nombre de succès")
```



3 Méthode de rejet

3.1 Simulation d'une loi de Poisson pour $\lambda < 1$

1. On utilisant le résultat de l'exercice 2.2, proposez un algorithme pour simuler une variable aléatoire de Bernouilli de paramètre $p \in]0, 1[$.

Il suffit de tirer une uniforme U, et de poser $X = 0$ si $U \leq 1 - p$, 1 sinon.

2. En déduire un algorithme pour simuler une loi géométrique de paramètre p sur $\mathbb{N}$.

Une variable aléatoire Y de loi géométrique sur $\mathbb{N}$ de paramètre p est la loi du nombre d'échecs d'une Bernouilli de paramètre p avant le premier succès.

Ainsi, on initialise à $Y = 0$, et on simule X. Tant que $X = 0$, on pose $Y = Y + 1$.

3. On souhaite obtenir un échantillon d'une loi de de Poisson de paramètre $\lambda \in]0, 1[$ par méthode d'acceptation rejet. On se propose d'utiliser comme loi de proposition la loi géométrique sur $\mathbb{N}$ de paramètre $1 - \lambda$. Définir l'algorithme de rejet correspondant.

On veut simuler selon Z, une variable aléatoire de loi de poisson de paramètre $\lambda \in]0, 1[$. On prend comme loi de proposition la loi géométrique sur $\mathbb{N}$ de paramètre $(1 - \lambda)$. On note Y une variable aléatoire de cette loi. Pour $k \in \mathbb{N}$, on a:

$$\mathbb{P}(Z = k) = \frac{\lambda^k}{k!} \mathrm{e}^{-\lambda}$$
$$\mathbb{P}(Y = k) = \lambda^k (1 - \lambda)$$

On a:

$$\frac{\mathbb{P}(Z = k)}{\mathbb{P}(Y = k)} = \frac{\mathrm{e}^{-\lambda}}{k!(1 - \lambda)}$$
$$\leq \frac{\mathrm{e}^{-\lambda}}{(1 - \lambda)} := M$$

Ainsi, pour simuler une réalisation z selon Z, on a l'algorithme suivant:

1. Simuler une réalisation y d'une loi géométrique de paramètre $p = 1 - \lambda$.
2. Simuler une loi uniforme u sur [0, 1].
3. $$u \leq \frac{\mathbb{P}(Z = y)}{M \mathbb{P}(Y = y)},$$
   poser $z = y$, sinon, retourner en 1.

4. Quelle est la probabilité d'acceptation dans l'algorithme de rejet?

La probabilité d'acceptation dans l'algorithme de rejet est donnée par $\frac{1-\lambda}{e^{-\lambda}}$, qui tend vers 0 quand λ tend vers 1.

5. Faites une fonction `R` permettant de générer une loi géométrique de paramètre p. Utiliser cette fonction dans une autre fonction `R` permettant de simuler selon une loi de Poisson de paramètre $p \in]0, 1[$. Simuler ainsi un échantillon de taille 10000. Comparer la distribution obtenue à celle de la vraie loi.

```r
# Simulation d'une bernouilli
mon_rbern <- function(n, p){
  us <- runif(n) # Echantillon uniforme
  ifelse(test = us < 1 - p, # Pour chaque u, on checke si u < 1- p
         yes = 0, # Si oui, on assigne la valeur 0
         no = 1)  # Si non la valeur 1
}
```

```r
# On peut donc coder la fonction rgeom qui simule une loi géométrique
# (défini comme le nombre d'échecs avant le 1er succès)
mon_rgeom <- function(n, p){
  get_one_sample <- function(){# On fait la fonction (en interne) pour 1 simu
    nb_echecs <- -1 # Nombre d'échecs courant (initialisé à - 1)
    stopping_condition <- FALSE
    while(!stopping_condition){
      nb_echecs <- nb_echecs + 1 # Le nombre d'échecs augmente
      bernouilli_sample <-  mon_rbern(1, p) # Simulation d'un Bernouilli
      stopping_condition <- bernouilli_sample == 1 # Condition d'arret
    }
    return(nb_echecs)
  }
  # On reproduit n fois le même traitement grâce à rerun
  rerun(n,
        get_one_sample()) %>% # Renvoit une liste
    unlist()  # Le resultat final est un vecteur
}
```

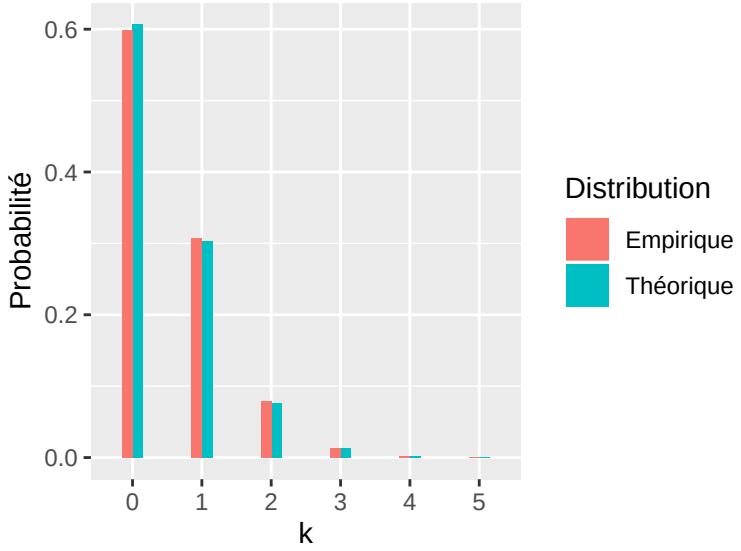
```r
# On se sert de cette fonction pour simuler une loi de poisson
mon_rpois <- function(n, lambda) {
  if (lambda >= 1 | lambda <= 0) {
    stop("In this function, lambda must be between 0 and 1")
  }
  M <- exp(-lambda) / (1 - lambda) # Borne uniforme
  get_one_sample <- function(){ # Fonction de génération d'un individu
    accepted <- FALSE # A t'on accepte un candidat?
    while(!accepted){
      candidate <- mon_rgeom(1, 1 - lambda) # Simule selon une géométrique
      # ratio d'acceptation
      # dpois et dgeom donnent les lois de poisson et geometrique (sur N) dans R
      ratio <- dpois(candidate, lambda) / (M * dgeom(candidate, 1 - lambda))
      u <- runif(1, min = 0, max = 1)
      accepted <- (u <= ratio)# A t'on accepté?
    }
    return(candidate)
  }
  rerun(n, # Rerun n times the get_one_sample function
                get_one_sample()) %>% # returns a list
    unlist() %>% # Transformed into a vector
    tibble(n = 1:n, x = .) # Put into a tibble to ease ggplots
}
```

```r
set.seed(123)# Manière de "fixer" l'aléa (pour des résultats reproductibles)
n_sample <- 1e4
lambda <- 0.5
poisson_sample <- mon_rpois(n_sample, lambda)
```

Pour la représentation graphique, on commence par extraire les fréquences empiriques et les fréquences théoriques.

```
# On représente l'histogramme empirique jusqu'à k_ma, le maximum observé
# dans l'échantillon
k_max <- max(poisson_sample$x)
frequences <- poisson_sample %>%
  mutate(x = factor(x, levels = 0:k_max)) %>% # On transforme en facteur
  group_by(x) %>%
  summarise(freq_empir = n() / n_sample) %>%
  ungroup() %>%
  mutate(freq_theo = dpois(0:k_max, lambda))
```

```
frequences %>% # On transforme légèrement pour le graphique
  gather(-x, key = "Distribution", value = "Proba") %>% # Regardez ce que ca fait!
  ggplot() + # on représente le résultat
  aes(x = x, y = Proba) + #
  geom_col(mapping = aes(fill = Distribution), # Ordonnée des fréquences empiriques
           width = 0.3, position = "dodge") +
  labs(x = "k", y = "Probabilité") +
  scale_fill_discrete(labels = c("Empirique", "Théorique"))
```



3.2 Loi uniforme sur le disque unité

1. À partir d'une variable aléatoire uniforme sur $[0, 1]$, proposez une transformation pour simuler une loi uniforme sur $[-1, 1]$.

Si U est une uniforme sur [0,1], alors, la variable aléatoire $2U - 1$ est une uniforme sur [-1, 1]

2. Proposer une méthode d'acceptation rejet pour simuler, à partir de deux variables aléatoires indépendantes de loi uniforme sur $[-1, 1]$, une variable aléatoire uniforme sur le disque unité.

Un point (x, y) de $\mathbb{R}^{\not\vDash}$ est dans le disque unité si $x^2 + y^2 \leq 1$. L'aire de cette surface est de π.

La densité de la loi uniforme sur le disque unité est donc

$$f_D(x, y) = \frac{\mathbf{1}_{x^2+y^2\leq 1}}{\pi}$$

La densité de la loi uniforme sur le carré de $[-1, 1] \times [-1, 1]$ est

$$g_C(x, y) = \frac{\mathbf{1}_{-1\leq x\leq 1}\mathbf{1}_{-1\leq y\leq 1}}{4}$$

On ainsi que pour tout (x, y) $f_D(x, y) \leq \frac{4}{\pi} g_C(x, y)$.

Le ratio d'acceptation est donc simplement

$$\alpha(x, y) = \frac{\pi}{4}\frac{f_D(x, y)}{g_C(x, y)} = \frac{\mathbf{1}_{x^2+y^2\leq 1}}{\mathbf{1}_{-1\leq x\leq 1}\mathbf{1}_{-1\leq y\leq 1}}$$

L'algorithme est donc:

1. On simule 2 uniformes U et V sur [-1, 1] indépendamment.
2. On accepte le couple U, V si il tombe dans le disque unité (i.e. $U^2 + V^2 \leq 1$), on recommence sinon

3. Quelle est la probabilité d'acceptation de l'algorithme?

La probabilité d'acceptation de l'algorithme est donc de $\frac{\pi}{4}$

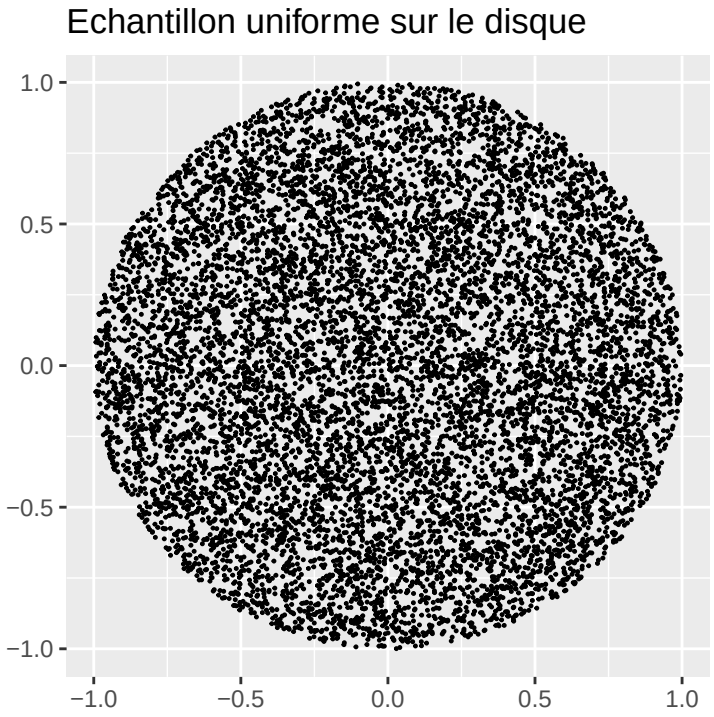
4. Ecrire une fonction `R` mettant en place la génération de variables aléatoires sur le disque unité. Dans cet algorithme, gardez en mémoire le nombre d'essais nécessaire avant chaque acceptation.

```r
sample_disc_uniform <- function(n){
  get_one_sample <- function(){
    n_essai = 0 # Nbre d'essai
    accepted = FALSE # Condition d'arret initialisee a FAUX
    while(!accepted){
      n_essai = n_essai + 1 # INcrementation
      candidate <- 2 * runif(2, 0, 1) - 1 #2 simulations indépendantes sur [-1, 1]
      # equivalent à runif(2, -1, 1)
      accepted <- sum(candidate^2) <= 1
    }
    # Attention, en R, l'indexation commence à 1
    tibble(x = candidate[1], y = candidate[2], n_essai = n_essai)
  }
  rerun(n, # Rerun n times the get_one_sample function
        get_one_sample()) %>% # returns a list of tibbles
    bind_rows(.id = "Replicate") # Bind it into a single tibble
}
```

5. Générer un échantillon de taille 10000. Vérifiez graphiquement que ces points sont uniformément répartis sur le disque unité. Vérifiez également que le nombre d'essais moyens avant acceptation est en adéquation avec ce qui est attendu.

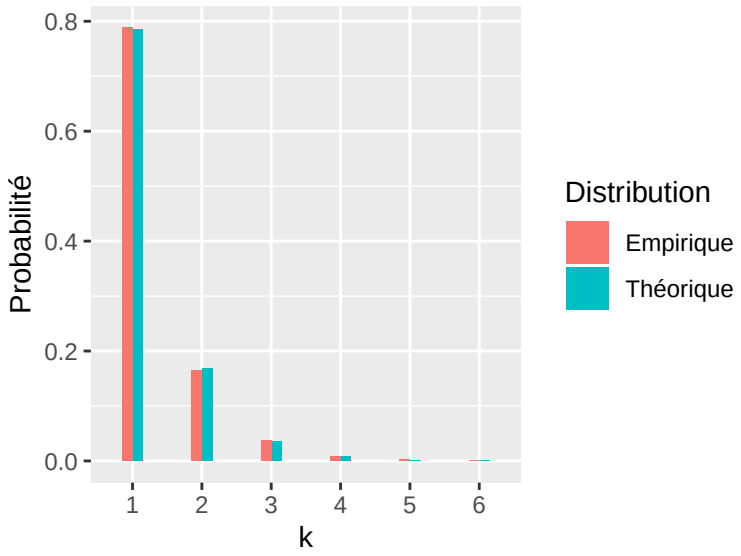
```r
set.seed(123) # Pour la reproductibilité
n_sample <- 1e4
disc_sample <- sample_disc_uniform(n_sample)
```

```r
ggplot(disc_sample, aes(x = x, y = y)) +
  geom_point(size = 0.2) +
  labs(x = "", y = "", title = "Echantillon uniforme sur le disque")
```



On peut regarder la distribution empirique du nombre d'essais moyens. La théorie stipule qu'il s'agit d'une loi géométrique sur $\mathbb{N}^*$ de paramètre $\frac{\pi}{4}$.

```r
k_max <- max(disc_sample$n_essai)
# D'abord en calcule les fréquences empiriques puis théoriques
disc_sample %>%
  mutate(n_essai = factor(n_essai, levels = 1:k_max)) %>% # On transforme en facteur
  group_by(n_essai) %>% # Regroupement
  summarise(freq_empir = n() / n_sample) %>%
  ungroup() %>% # Degroupement
  mutate(freq_theo = dgeom(0:(k_max - 1), pi / 4)) %>% # Dans R, la loi geometrique
  # est sur N, donc on retranche 1.
  # Puis on transforme légèrement pour le graphique
  gather(-n_essai, key = "Distribution", value = "Proba") %>% # Regardez ce que ca fait!
  ggplot() + # on représente le résultat
  aes(x = n_essai, y = Proba) + #
  geom_col(mapping = aes(fill = Distribution), # Ordonnée des fréquences empiriques
           width = 0.3, position = "dodge") +
  labs(x = "k", y = "Probabilité") +
  scale_fill_discrete(labels = c("Empirique", "Théorique"))
```



3.3 Proposition optimale

La loi normale tronquée de support $[b, +\infty[$ est définie par la densité f proportionnelle, pour tout réel x, à

$$f_1(x) = \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}\mathbf{1}_{x\geq b}, \quad \text{avec } \mu > 0,\ \sigma > 0.$$

On propose de simuler suivant la loi de densité f par une méthode de rejet.

3.3.1 Méthode naïve

1. On note Φ la fonction de répartition de la loi normale centrée réduite. Montrer que f satisfait l'inégalité suivante pour tout réel x:
$$f(x) \leq \frac{1}{\sigma\sqrt{2\pi}\phi(\frac{\mu-b}{\sigma})}\exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}.$$

On a que $f(x) \propto f_1(x)$. Donc

$$f(x) = \frac{f_1(x)}{\int_b^\infty f_1(z)\mathrm{d}z}$$

Or:

$$\begin{aligned}\int_b^\infty f_1(z)\mathrm{d}z &= \sqrt{2\pi\sigma^2}\int_b^\infty \frac{1}{\sqrt{2\pi\sigma^2}}\mathrm{e}^{-\frac{(z-\mu)^2}{2\sigma^2}} \\ &= \sqrt{2\pi\sigma^2}\mathbb{P}\left(\mu + \sigma Z > b\right)\end{aligned}$$

où $Z \sim \mathcal{N}(0,1)$

$$= \sqrt{2\pi\sigma^2}\mathbb{P}\left(Z > \frac{b-\mu}{\sigma}\right)$$

$$= \sqrt{2\pi\sigma^2}\mathbb{P}\left(Z < \frac{\mu-b}{\sigma}\right)$$

Donc

$$f(x) \leq \frac{1}{\sigma\sqrt{2\pi}\phi(\frac{\mu-b}{\sigma})} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\} \mathbf{1}_{x\geq b},$$

qui satisfait bien l'inéquation voulue.

2. En déduire l'algorithme du rejet. Que peut-on dire du nombre d'essais moyen avant acceptation ?

Donc, si on fait un algorithme de rejet avec commme loi de proposition g la densité d'une Gaussienne de moyenne μ et variance σ^2. On a alors:

$$\frac{f(x)}{g(x)} \leq \frac{1}{\phi(\frac{\mu-b}{\sigma})}$$

La probabilité d'acceptation est donc exactement $\phi(\frac{\mu-b}{\sigma})$. Ainsi, si

- $b < \mu$ alors, la probabilité d'acceptation est supérieure à 0.5.
- Si $b >> \mu$, alors la probabilité d'acceptation est très petite!

3.3.2 Une distribution instrumentale alternative

On suppose que $b > \mu$. On considère la loi exponentielle translatée de b, $\tau\mathcal{E}(\lambda, b)$, de densité

$$g_\lambda(x) = \lambda e^{-\lambda(x-b)} \mathbf{1}_{x\geq b}, \quad x \in \mathbb{R}.$$

3. Montrer pour $x \geq b$ que

$$\frac{f_1(x)}{g_\lambda(x)} \leq \begin{cases} \frac{1}{\lambda} \exp\left\{\lambda(\mu-b) + \frac{(\lambda\sigma)^2}{2}\right\} & \text{si } \mu + \lambda\sigma^2 > b, \\ \frac{1}{\lambda} \exp\left\{-\frac{(b-\mu)^2}{2\sigma^2}\right\} & \text{sinon.} \end{cases}$$

Soit $x \geq b$:

$$\begin{aligned} \frac{f_1(x)}{g(x)} &= \frac{1}{\lambda} \mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2} + \lambda(x-b)} \\ &= \frac{1}{\lambda} \mathrm{e}^{-\frac{1}{2\sigma^2} \overbrace{\left((x-\mu)^2 - 2\sigma^2\lambda(x-b)\right)}^{:=P(x)}} \end{aligned}$$

où

$$\begin{aligned} P(x) &= x^2 - 2\mu x - 2\lambda\sigma^2 x + \mu^2 + 2\lambda\sigma^2 b \\ &= \left(x - (\mu + \lambda\sigma^2)\right)^2 - (\mu + \lambda\sigma^2)^2 + \mu^2 + 2\lambda\sigma^2 b \\ &= \left(x - (\mu + \lambda\sigma^2)\right)^2 - 2\sigma^2 \left(\lambda(\mu - b) + \frac{\lambda^2\sigma^2}{2}\right). \end{aligned}$$

On travaille sur l'ensembke $x \geq b$. Ainsi, si $\mu + \lambda\sigma^2 > b$ $P(x)$ atteint son minimum en $x = \mu + \lambda\sigma^2$. et le maximum du ratio est atteint en cette valeur. Si $\mu + \lambda\sigma^2 \leq b$, $P(x)$ est minimal (donc le ratio maximal), en $x = b$. On retrouve donc l'expression cherchée.

4. Proposer une méthode de simulation de la loi de densité f.

Ainsi à $b, \mu, et \sigma^2$ fixés, il suffit, pour un λ choisi,

1. De calculer $M(\lambda)$ comme une des deux valeurs données plus haut, selon que $\mu + \lambda\sigma^2 > b$ ou non.
2. De simuler Y comme une exponentielle de paramètre λ, à laquelle on ajoute b.
3. Simuler une uniforme $U \sim \mathcal{U}[0,1]$;
4. Si $U < \frac{f_1(Y)}{M(\lambda)g(Y)}$, alors on pose $X = Y$, sinon, on recommence.

5. Calculer la valeur de λ^* telle que le temps moyen de calcul de la méthode proposée soit le plus petit possible.

On veut minimiser:

$$M(\lambda) = \begin{cases} \frac{1}{\lambda}\exp\left\{\lambda(\mu-b) + \frac{(\lambda\sigma)^2}{2}\right\} & \text{si } \lambda > \frac{b-\mu}{\sigma^2}, \\ \frac{1}{\lambda}\exp\left\{-\frac{(b-\mu)^2}{2\sigma^2}\right\} & \text{si } 0 < \lambda \leq \frac{b-\mu}{\sigma^2}. \end{cases}$$

Sur l'intervalle $]0, \frac{b-\mu}{\sigma^2}]$, $M(\lambda)$ est décroissante, le minimum est donc atteint $\frac{b-\mu}{\sigma^2}$. On notera que $M(\lambda)$ est continue en ce point, et donc continue partout.

Sur $]\frac{b-\mu}{\sigma^2}, +\infty[$, on a

$$M'(\lambda) = \frac{1}{\lambda}M(\lambda)\left(\sigma^2\lambda^2 + \lambda(\mu-b) - 1\right)$$

Ainsi, la dérivée est négative entre les deux solutions du polynôme qui sont:

$$\lambda_1 = \frac{(b-\mu) - \sqrt{(b-\mu)^2 + 4\sigma^2}}{2\sigma^2} < 0$$

$$\lambda_2 = \frac{(b-\mu) + \sqrt{(b-\mu)^2 + 4\sigma^2}}{2\sigma^2} > \frac{b-\mu}{\sigma^2}$$

Ainsi, le minimum global de $M(\lambda)$ est atteint en λ_2. On a donc $\lambda^* = \lambda_2$.

6. En R, mettre en oeuvre cette dernière méthode.

```
# On définit les fonctions apparaissant
f1 <- function(xs, b, mu, sigma2){
  ifelse(test = xs >= b,
         yes = exp(-0.5 * (xs - mu)^2 / sigma2),
         no = 0)
}

target_f <- function(xs, b, mu, sigma2){
  constante_normalisation <- sqrt(2 * pi * sigma2) * pnorm((mu - b) / sigma2)
  f1(xs, b, mu, sigma2) / constante_normalisation
}

g_lambda <- function(xs, b, lambda){
  ifelse(xs > b, lambda * exp(-lambda * (xs - b)), 0)
}

get_lambda_opt <- function(b, mu, sigma2){
  0.5 * (b - mu + sqrt((b - mu)^2 + 4 * sigma2)) / sigma2
```

```
}

get_M_lambda <- function(lambdas, b, mu, sigma2){
  ifelse(lambdas > ((b - mu) / sigma2),
         yes = exp(lambdas * (mu - b) + 0.5 * sigma2 * lambdas^2 ) / lambdas,
         no = exp(-0.5 * (b - mu)^2 / sigma2) / lambdas)
}
```

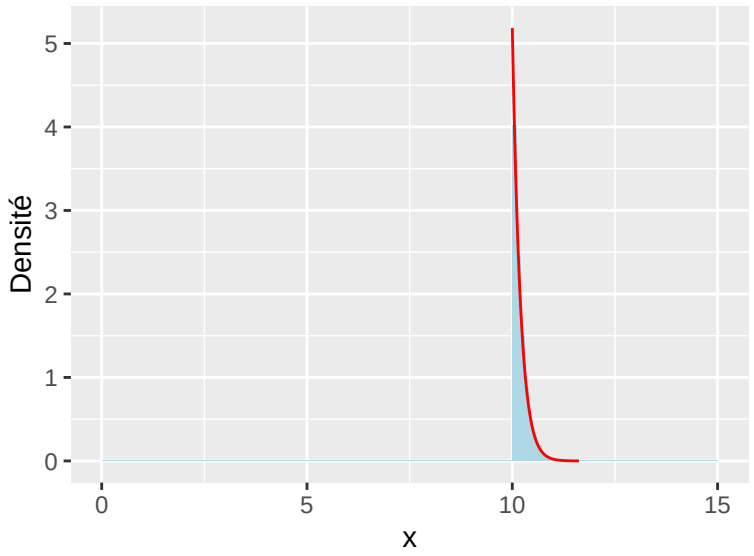
Avec ces fonctions, on peut définir l'algorithme de simulation

```
mon_random_f <- function(n, b, mu, sigma2){
  lambda_opt <- get_lambda_opt(b, mu, sigma2)# On calcule le lambda optimal
  M <- get_M_lambda(lambda_opt, b, mu, sigma2)# Le M de l'acceptation rejet
  get_one_sample <- function(){
    n_essai <- 0
    accepted <- FALSE
    while(!accepted){
      n_essai = n_essai + 1
      candidat <- b + rexp(1, lambda_opt) # Exponentielle translatee
      ratio <- f1(candidat, b, mu, sigma2) / (M * g_lambda(candidat, b, lambda_opt))
      u <- runif(1)
      accepted <- (u <= ratio)
    }
    tibble(x = candidat, n_essai = n_essai)
  }
  # On répète n_fois la même fonction, qui renvoie un tibble
  # tous les résultats sont ensuite concaténés
  res <- purrr::rerun(n, get_one_sample()) %>%#
    dplyr::bind_rows() # Concatenation
  return(res)
}
```

Que l'on fait tourner pour certaines valeurs de b, μ et σ^2

```
b <- 10; mu <- 5; sigma2 <- 1 # paramètres
trunc_normal_samples <- mon_random_f(10000, b, mu, sigma2)
```

```
my_breaks <- c(0, seq(b, b + mu, length.out = 51))
ggplot(trunc_normal_samples, aes(x = x)) +
  geom_histogram(aes(y = ..density..), breaks = my_breaks,
                 fill = "lightblue") +
  stat_function(fun = function(x) target_f(x, b, mu, sigma2), color = "red") +
  labs(y = "Densité")
```



On verra dans l'exercice de la section 5 que la probabilité d'acceptation peut être calculée ici. On peut aussi calculer la proba de l'algo naif.

```
(proba_naive <- pnorm((mu - b) / sigma2))# Proba de l'algo naif
```

```
[1] 2.866516e-07
```

```
1 / mean(trunc_normal_samples$n_essai)# Proba empirique de l'algo optimisé
```

```
[1] 0.9821253
```

```
# On peut en fait connaitre aussi la proba theorique (exo suivant)
lambda_star <- get_lambda_opt(b, mu, sigma2)
M_star <- get_M_lambda(lambda_star, b, mu, sigma2)
(proba_theo <- sqrt(2 * pi * sigma2) * proba_naive / M_star)
```

```
[1] 0.9827773
```

4 Méthode de transformation

4.1 Simulation de lois Gaussiennes. Algorithme de Box-Muller

Soient X et Y deux variables aléatoires indépendantes de loi $\mathcal{N}(0,1)$

1. Montrer que si U et V sont deux variables aléatoires indépendantes de loi $\mathcal{U}[0,1]$ alors le couple

$$\left(\sqrt{-2\ln(U)}\cos(2\pi V), \sqrt{-2\ln(U)}\sin(2\pi V)\right)$$

   a la même loi que le couple (X,Y).

Pour $(u,v) \in]0,1[^2$, on note

$$\phi(u,v) = \left(\sqrt{-2\ln(u)}\cos(2\pi v), \sqrt{-2\ln(u)}\sin(2\pi v)\right)$$

On peut remarquer que pour tout $(x,y) \in \mathbb{R}_*^2$, la fonction définie par

$$\phi^{-1}(x,y) = \begin{cases} \left(\exp(-(x^2+y^2)/2), \frac{\arctan(y/x)}{2\pi}\right) & \text{si } x>0,\ y\geq 0 \\ \left(\exp(-(x^2+y^2)/2), \frac{\arctan(y/x)+2\pi}{2\pi}\right) & \text{si } x>0,\ y<0 \\ \left(\exp(-(x^2+y^2)/2), \frac{\arctan(y/x)+\pi}{2\pi}\right) & \text{si } x<0 \\ \left(\exp(-(x^2+y^2)/2), \frac{1}{4}\right) & \text{si } x=0, y>0 \\ \left(\exp(-(x^2+y^2)/2), \frac{3}{4}\right) & \text{si } x=0, y<0 \end{cases}$$

est l'inverse de ϕ. Ainsi, si tout les intervalles où ϕ^{-1} Ainsi, sur tout intervalle où cette fonction est différentiable, on peut écrire la Jacobienne

$$J_{\phi^{-1}}(x,y) = \begin{pmatrix} \frac{\delta\phi_1^{-1}}{\delta x}(x,y) & \frac{\delta\phi_1^{-1}}{\delta y}(x,y) \\ \frac{\delta\phi_2^{-1}}{\delta x}(x,y) & \frac{\delta\phi_2^{-1}}{\delta y}(x,y) \end{pmatrix} = \begin{pmatrix} -x\mathrm{e}^{-\frac{1}{2}(x^2+y^2)} & -y\mathrm{e}^{-\frac{1}{2}(x^2+y^2)} \\ \frac{1}{2\pi}\frac{-y}{x^2+y^2} & \frac{1}{2\pi}\frac{x}{x^2+y^2} \end{pmatrix}.$$

Ainsi, par formule du changement de variable, la densité $f_{X,Y}(x,y)$ du couple de variables aléatoires (X,Y) est donnée par:

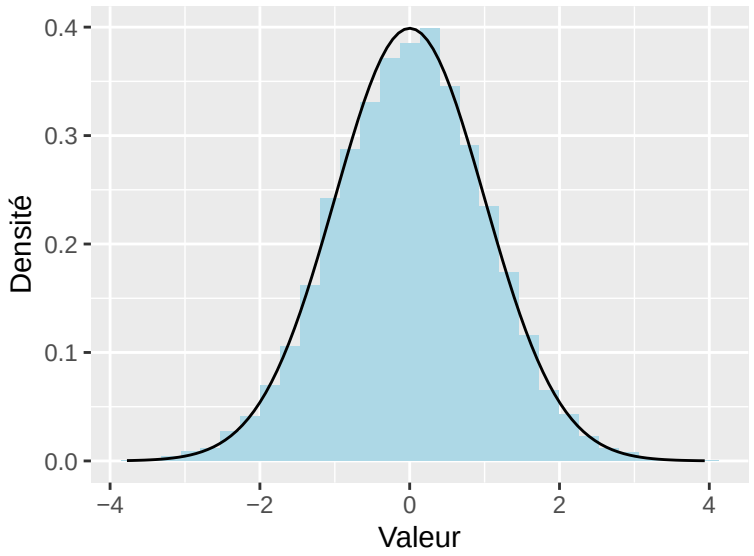
$$f_{X,Y}(x,y) = f_{U,V}(\phi^{-1}(x,y))|\det J_{\phi^{-1}}(x,y)| = \frac{1}{2\pi}\mathrm{e}^{-\frac{1}{2}(x^2+y^2)} = \frac{1}{\sqrt{2\pi}}\mathrm{e}^{-\frac{1}{2}x^2}\frac{1}{\sqrt{2\pi}}\mathrm{e}^{-\frac{1}{2}y^2}$$

On peut ainsi prolonger par continuité la fonction $f_{X,Y}$ en tous les points de discontinuité de ϕ^{-1}. Le couple (X,Y) est donc un couple de 2 variables aléatoires indépendantes, de loi $\mathcal{N}(0,1)$.

2. Ecrire une fonction `box_muller` permettant de simuler une loi $\mathcal{N}(0,1)$ en `R`. Vous comparerez l'histogramme obtenu à la vrai densité de la loi.

```
box_muller <- function(n_sample){
  n_uniform_sample <- ceiling(0.5 * n_sample)
  u <- runif(n_uniform_sample)
  v <- runif(n_uniform_sample)
  first_half <- sqrt(-2 * log(u)) * cos(2 * pi * v)
  second_half <- sqrt(-2 * log(u)) * sin(2 * pi * v)
  tibble(sample = c(first_half, second_half))
}
```

```
box_muller(1e4) %>%
  ggplot(aes(x = sample)) +
  geom_histogram(aes(y = ..density..), fill = "lightblue") +
  stat_function(fun = dnorm) +
  labs(x = "Valeur", y = "Densité")
```



3. En déduire, pour tout $\mu \in \mathbb{R}^{\not\vdash}$ et toute matrice 2×2 symmétrique semi-définie positive Σ une méthode pour simuler une variable aléatoire $Z \sim \mathcal{N}(\mu, \Sigma)$.

Pour toute matrice semi définie positive Σ, on peut définir une unique matrice triangulaire inférieure, notée $\Sigma^{\frac{1}{2}}$, telle que $\Sigma^{\frac{1}{2}}(\Sigma^{\frac{1}{2}})^T = \Sigma$ (cette matrice est la transposée de la décomposition de Choleski). De plus, on rappelle que si $X \sim \mathcal{N}(0, I)$, alors, pour un vecteur a et une matrice B de dimensions adéquates $\mu + BX \sim \mathcal{N}(\mu, BB^T)$. À partir de l'algorithme de Box-Muller, on peut donc simuler n'importe quel vecteur Gaussien.

5 Autour de l'accepation rejet

5.1 Acceptation rejet étendu: Cas de deux fonctions positives.

Pour cette preuve, vous pourrez mimer les étapes de la preuve dans le cas usuel, détaillée dans le poly.

On se propose de montrer que pour que simuler selon une densité par algorithme d'acceptation rejet, il n'est nécessaire de connaître la densité qu'*à la constante de normalisation près.* Cette propriété est très utile dans le cas où le calcul de la constante de normalisation est coûteux, voir impossible (typiquement en statistiques Bayesiennes).

Plus formellement, soient $\tilde{f}$ une fonction positive et g une densité de probabilité, toutes deux définies sur $\mathbb{R}^d$ telles que:

- $0 < \int_{\mathbb{R}^d} \tilde{f}(x)\mathrm{d}x < \infty$. On note respectivement $I(\tilde{f})$ cette intégrale et

$$f(x) = \frac{\tilde{f}(x)}{I(\tilde{f})}$$

  la densité associée à cette fonction positive.
- Il existe $M > 0$ tel que, pour tout réel x, $\tilde{f}(x) \leq Mg(x)$.

On note

$$\alpha(x) := \frac{\tilde{f}(x)}{Mg(x)}.$$

Soit $(U_m)_{m\geq 1}$ une suite de variables aléatoires i.i.d. de loi uniforme sur $[0, 1]$. Soit $(Y_m)_{m\geq 1}$ une suite de variables aléatoires indépendantes et identiquement distribuée, de densité donnée par g. On note T la

variable aléatoire (à valeurs dans $\mathbb{N}^*$):

$$T = \inf\{m, \text{ tel que } U_m \leq \alpha(Y_m)\}.$$

.

1. Montrer la variable aléatoire $X := Y_T$ (T-ième valeur de la suite $(Y_m)_{m\geq 1}$) a pour densité f.

On reprend, à la normalisation près, la démonstration du cours:

Soit un entier $n \leq 1$:

$$\mathbb{P}(X \leq x, T = n) = \mathbb{P}(U_1 > \alpha(Y_1), \ldots, U_{n-1} > \alpha(Y_{n-1}), U_n \leq \alpha(Y_n), Y_n \leq x)$$

Par propriété d'indépendance

$$= \mathbb{P}(U_n \leq \alpha(Y_n), Y_n \leq x) \prod_{i=1}^{n-1} \mathbb{P}(U_i > \alpha(Y_i))$$

Par propriété de distribution identique

$$= \mathbb{P}(U_n \leq \alpha(Y_n), Y_n \leq x)\, \mathbb{P}(U_1 > \alpha(Y_1))^{n-1}$$

Or on a:

$$\begin{aligned}
\mathbb{P}(U_1 > \alpha(Y_1)) &= \mathbb{E}\left[\mathbf{1}_{U_1 > \alpha(Y_1)}\right] \\
&= \int_{\mathbb{R}} \left(\int_0^1 \mathbf{1}_{u > \frac{\tilde{f}(y)}{Mg(y)}} \mathrm{d}u \right) \times g(y) \mathrm{d}y \\
&= \int_{\mathbb{R}} \left(1 - \frac{I(\tilde{f}) f(y)}{Mg(y)}\right) g(y) \mathrm{d}y
\end{aligned}$$

Comme f est une densité:

$$\mathbb{P}(U_1 > \alpha(Y_1)) = 1 - \frac{I(\tilde{f})}{M}$$

De manière analogue

$$\begin{aligned}
\mathbb{P}(U_n \leq \alpha(Y_n), Y_n \leq x) &= \mathbb{E}\left[\mathbf{1}_{U_n \leq \alpha(Y_n)} \times \mathbf{1}_{Y_n \leq x}\right] \\
&= \int_{\mathbb{R}} \left(\int_0^1 \mathbf{1}_{u \leq \frac{f(y)}{Mg(y)}} \mathrm{d}u \right) \times \mathbf{1}_{y \leq x} g(y) \mathrm{d}y \\
&= \int_{-\infty}^{x} \frac{I(\tilde{f}) f(y)}{M} \mathrm{d}y \\
&= \frac{I(\tilde{f}) F(x)}{M},
\end{aligned}$$

où $F(x)$ est la fonction de répartition associée à f. En résumé:

$$\mathbb{P}(X \leq x, T = n) = \frac{I(\tilde{f}) F(x)}{M} \left(1 - \frac{I(\tilde{f})}{M}\right)^{n-1}$$

Donc on conclut en remarquant que

$$\mathbb{P}(X \leq x) = \sum_{n=1}^{\infty} \mathbb{P}(X \leq x, T = n) = F(x)$$

2. Donnez alors la loi de la variable aléatoire T. Quelle est l'espérance de T?

De la preuve, on peut déduire que

$$\mathbb{P}(T = n) = \lim_{x \to \infty} \mathbb{P}(X \leq x, T = n) = \lim_{x \to \infty} \frac{I(\tilde{f})F(x)}{M} \left(1 - \frac{I(\tilde{f})}{M}\right)^{n-1} = \frac{I(\tilde{f})}{M} \left(1 - \frac{I(\tilde{f})}{M}\right)^{n-1}$$

Donc, la loi de T est une loi géométrique sur $\mathbb{N}_*$ de paramètre $\frac{I(\tilde{f})}{M}$. L'espérance de T est donc donnée par $\frac{M}{I(\tilde{f})}$.

3. En déduire un estimateur de $I(\tilde{f})$ par méthode de Monte Carlo, obtenu uniquement à partir de l'algorithme d'acceptation rejet défini plus tôt.

Supposons qu'on simule un échantillon $X_1, \ldots, X_n$ de variables aléatoires i.i.d. de densité f par l'algorithme d'acceptation rejet. Pour chaque $X_i (1 \leq i \leq n)$, on note T_i le temps d'arrêt correspondant.

Ainsi, l'estimateur

$$\hat{\tau}_n = \frac{1}{n} \sum_{i=1}^{n} T_i$$

est un estimateur sans biais de $\frac{M}{I(\tilde{f})}$. Donc, M étant connu,

$$\hat{I}_n(\tilde{f}) = \frac{M}{\hat{\tau}_n}$$

est un estimateur consistant de $I(\tilde{f})$.

4. Grâce au théorème central limite, donnez l'expression d'un intervalle de confiance asymptotique à 95% pour $I(\tilde{f})$, ne dépendant d'aucune quantité inconnue.

Le théorème central limite nous donne:

$$\sqrt{n} \left(\hat{\tau}_n - \frac{M}{I(\tilde{f})}\right) \xrightarrow{loi} \mathcal{N}(0, \sigma^2)$$

où $\sigma^2 = \mathbb{V}[\hat{\tau}_n]$. Par la Δ- méthode, on a que

$$\sqrt{n} \left(\hat{I}_n(\tilde{f}) - I(\tilde{f})\right) \xrightarrow{loi} \mathcal{N}(0, \tilde{\sigma}^2)$$

où $\tilde{\sigma} = \frac{M^2}{\hat{\tau}_n^4} \sigma^2$

Or un estimateur consistant de σ^2 est

$$S^2 = \frac{1}{n} \sum_{i=1}^{n} (T_i - \hat{\tau}_n)^2$$

Donc

$$\frac{1}{n}\frac{M^2}{\hat{\tau}_n^4}S^2$$

est un estimateur consistant de la variance de $\hat{I}_n(\tilde{f})$.

Un intervalle de confiance asymptotique à 95% est donc donné par:

$$\left[\hat{I}_n - 1.96\frac{M}{\hat{\tau}_n^2}\sqrt{\frac{S^2}{n}}; \hat{I}_n + 1.96\frac{M}{\hat{\tau}_n^2}\sqrt{\frac{S^2}{n}}\right]$$

5.2 Recyclage dans l'acceptation rejet

Dans cette section, on se replace dans le cadre classique de l'acceptation rejet.

On se propose d'approcher une intégrale du type: $J = \mathbb{E}_f[\varphi(X)]$ où $f($ est la densité de la variable aléatoire X sur $\mathbb{R}^d$ selon laquelle on ne sait pas simuler, et φ est une fonction intégrable par rapport à cette densité.

À partir d'une densité $g(x)$ sur $\mathbb{R}^d$ selon laquelle on sait simuler, et telle que

$$\exists M > 0, \text{ tel que } \forall x \in \mathbb{R}^d, \ f(x) \leq Mg(x)$$

on obtient, par algorithme d'acceptation-rejet (pour un tel M fixé) un échantillon de variables aléatoires *i.i.d.* $X_1, \ldots, X_n$ de loi donnée par f.

Pour obtenir cet échantillon de taille n, on a simulé $N \geq n$ variables aléatoires indépendantes $Y_1, \ldots, Y_N$ de densité g. On note $Z_1, \ldots, Z_{N-n}$ l'échantillon i.i.d. de variables aléatoires ayant été rejetées dans l'algorithme d'acceptation rejet.

5. Donner l'expression de la densité de la variable aléatoire Z_1.

$$\begin{aligned}\mathbb{P}(Z_1 \leq z) &= \mathbb{P}\left(Y_1 \leq z | U > \alpha(Y_1)\right) \\ &= \frac{\mathbb{P}\left(Y_1 \leq z, U > \alpha(Y_1)\right)}{\mathbb{P}\left(U > \alpha(Y_1)\right)}\end{aligned}$$

Le dénominateur est égal à $1 - \frac{1}{M}$ (voir section précédente, ou le cours). Intéressons nous au numérateur

$$\begin{aligned}\mathbb{P}\left(Y_1 \leq z, U > \alpha(Y_1)\right) &= \mathbb{E}\left[\mathbf{1}_{Y_1 \leq z}\mathbf{1}_{U > \alpha(Y_1)}\right] \\ &= \int_{\mathbb{R}}\int_0^1 \mathbf{1}_{u > \alpha(y)}\mathrm{d}u\mathbf{1}_{y \leq z}g(y)\mathrm{d}y \\ &= \int_{\mathbb{R}}\left(1 - \frac{f(y)}{Mg(y)}\right)\mathbf{1}_{y \leq z}g(y)\mathrm{d}y \\ &= \int_{-\infty}^{z} g(y) - \frac{f(y)}{M}dy\end{aligned}$$

Ainsi, on a

$$\mathbb{P}(Z_1 \leq z) = \frac{M}{M-1}\int_{-\infty}^{z} g(y) - \frac{f(y)}{M}dy = \int_{-\infty}^{z}\frac{Mg(y) - f(y)}{M-1}\mathrm{d}y$$

La densité de Z_1 est donc $h(z) = \frac{Mg(z) - f(z)}{M-1}$

6. En déduire que

$$\hat{J}_n = \frac{1}{N}\left(\sum_{i=1}^{n}\varphi(X_i) + \sum_{j=1}^{N-n}\frac{(M-1)f(Z_j)}{Mg(Z_j)-f(Z_j)}\varphi(Z_j)\right)$$

est un estimateur sans biais de J. Quelle est l'intérêt de cette méthode selon vous?

On s'intéresse $\mathbb{E}[\hat{J}_N]$. Il faut ici remarquer que N est également aléatoire. On passera par l'espérance conditionnelle

$$\mathbb{E}\left[\hat{J}_n\right] = \mathbb{E}\left[\mathbb{E}\left[\hat{J}_n|N\right]\right] = \mathbb{E}\left[\frac{1}{N}\left(\sum_{i=1}^{n}\mathbb{E}[\varphi(X_i)|N] + \sum_{j=1}^{N-n}\mathbb{E}\left[\frac{(M-1)f(Z_j)}{Mg(Z_j)-f(Z_j)}\varphi(Z_j)\middle| N\right]\right)\right]$$

Chacune des espérances conditionnelles est égale J (le premier, car c'est un estimateur Monte Carlo standard, le deuxième car c'est un estimateur par échantillonage préférentiel quand la loi de proposition est h!). Ainsi, $\hat{J}_n$ est bien d'espérance J.

Cette méthode permet donc d'avoir un estimateur sans biais de J à partir de tous les échantillons simulés pour l'acceptation rejet.

5.3 Application

On reprend l'exemple vu en cours d'introduction à la statistique bayésienne. Vous reprendrez le même modèle ainsi que les mêmes données utilisées.

7. En utilisant le même prior que celui du cours, ainsi que le même loi de proposition g, implémentez l'algorithme d'acceptation rejet pour tirer selon le posterior. Les algorithmes efficaces seront valorisés.

On reprend les notations du cours (voir poly ou diapos)

Pour un échantillon observé $y_{1:n} = y_1, \ldots, y_n$ et les covariables associées $\mathbf{x}_1, \ldots, \mathbf{x}_n$, on pose un modèle probit. La vraisemblance est données par

$$L(y_{1:n}|\theta, \mathbf{x}_{1:n}) = \prod_{k=1}^{n} \underbrace{\phi(\mathbf{x}_k^T\theta)^{y_k}}_{\text{Proba. présence}} \times \underbrace{(1-\phi(\mathbf{x}_k^T\theta))^{1-y_k}}_{\text{Proba. absence}}$$

où ϕ est la fonction de répartition d'une loi $\mathcal{N}(0,1)$.

Sur les paramètres inconnus, on pose un prior $\pi(\theta) \sim \mathcal{N}(0, 4I_4)$. Le posterior est donc donné par:

$$\pi(\theta|y_{1:n}, \mathbf{x}_{1:n}) \propto \pi(\theta)L(y_{1:n}|\theta, \mathbf{x}_{1:n}) \propto \frac{1}{64\pi^2}\mathrm{e}^{-\frac{1}{8}\theta^T\theta}\prod_{k=1}^{n}\phi(\mathbf{x}_k^T\theta)^{y_k}(1-\phi(\mathbf{x}_k^T\theta))^{1-y_k}$$

On note

$$\tilde{\pi}(\theta|y_{1:n}, \mathbf{x}_{1:n})) = \frac{1}{64\pi^2}\mathrm{e}^{-\frac{1}{8}\theta^T\theta}\prod_{k=1}^{n}\phi(\mathbf{x}_k^T\theta)^{y_k}(1-\phi(\mathbf{x}_k^T\theta))^{1-y_k}$$

Afin de simuler cette loi par acceptation rejet, on choisit comme loi de proposition la loi du prior (i.e. $g(\theta) = \pi(\theta)$). On remarque ainsi que pour tout θ,

$$\frac{\tilde{\pi}(\theta|y_{1:n}, \mathbf{x}_{1:n}))}{g(\theta)} = L(y_{1:n}|\theta, \mathbf{x}_{1:n})$$

De par l'expression de la vraisemblance, un majorant naturel, noté $\tilde{M}$ de ce ratio est 1.

Cependant, un majorant plus efficace est donné par la valeur du maximum de vraisemblance. Cette valeur est celle calculée pour effectuer l'inférence "classique".

Ainsi, en trouvant cette valeur, on obtiendra un $\tilde{M}$ qui sera plus petit, donc la probabilité d'acceptation (donnée en question 2.) sera plus grande.

Notre algorithme pour obtenir un échantillon du posterior est donc le suivant:

1. Trouver $\tilde{M} = \max_\theta L(y_{1:n}|\theta, \mathbf{x}_{1:n})$
2. Simuler un candidat $\theta_{cand} \sim \mathcal{N}(0, 4I_4)$
3. Simuler (indépendemment de U) $U \sim \mathcal{U}[0, 1]$.
4. Si $U \leq \frac{L(y_{1:n}|\theta_{cand}, \mathbf{x}_{1:n})}{\tilde{M}}$, on accepte θ_{cand}, sinon, on retourne à l'étape 2.

8. Implémentez cette méthode, et tracez les densités empiriques des posteriors obtenus. Vous donnerez également une estimation de $\mathbb{E}[\theta|y_{1:n}]$ ainsi que l'intervalle de confiance associé.

La première chose à faire est d'écrire la fonction de vraisemblance. Elle prendra en entrée un vecteur de β, une matrice X qui est la matrice de design, un vecteur y de 0 et de 1, et un argument log qui sera utile pour éviter les problèmes numériques.

```
get_likelihood <- function(beta_vec,
                           X, # Design matrix
                           y, # presence absence vector (numeric with 0 or 1)
                           return_log = FALSE){ # Return the loglikelihood, default: No
  X_beta <- as.numeric(X %*% beta_vec) # The as.numeric returns a vector
  log_phi <- pnorm(X_beta, log.p = T) # Evaluates the phi. Taking the log
  # is a usual trick to avoid numerical problems
  # Ifelse vectorise binary tests
  log_probability <- ifelse(test = (y == 1), # Test if presence
                            yes = log_phi, # Then the proba is phi(y)
                            no = log(1 - exp(log_phi))) # Else 1 - phi
  # Then, return either the result or its exponential (depending on return_log)
  ifelse(return_log,
         yes = return(sum(log_probability)),
         no = return(exp(sum(log_probability))))
}
```

On définit ensuite proprement la matrice de design ainsi que le vecteur de presence absence à partir des données

```
library(tidyverse)
# Loading data
donnees_presence <- read.table("https://papayoun.github.io/courses/2020_monte_carlo/enonces_td/donnees_p
                               sep = ";", header = TRUE,
                               colClasses = c(presence = "factor"))
# Creating design matrix
design_matrix <- donnees_presence %>% # A partir des données
  select(-presence) %>% # On vire la colonne des presence
  as.matrix() %>% # On transforme en matrice
  cbind(intercept = rep(1, nrow(.)), .) # On colle à gauche une colonne
# remplie de 1 pour l'intercept
y_vector <- donnees_presence %>%
  pull(presence) %>% # On extrait la colonne presence (de type factor)
```

```
  as.character() %>% # on transforme en string
  as.numeric() # on transforme en numérique
```

Détermination de $\tilde{M}$ par optimisation de la vraisemblance

Optimiser la vraisemblance du modèle probit est un problème classique en statistiques. La plupart des logiciels le font dans des fonctions dédiées (la fonction `glm` dans R, par exemple). Ici, on peut utiliser une fonction d'optimisation numérique donnée par R (la fonction `optim`) pour le faire à notre place, de manière efficace. Afin de savoir utiliser cette fonction, on se réfèrera à `help(optim)`.

Remarque: Ce cours n'étant pas un cours d'optimisation numérique, il n'est pas nécessaire de coder sa fonction soit même, ce qui serait fastidieux.

```
# On optimise par un algo de Nelder-Mead
# La fonction optim trouve le minimum d'une fonction
# Donc on veut minimiser -log_vraisemblance
M_tilde <- optim(par = rep(0, 4), # point de depart de l'algorithm
                 fn = function(b){ # Fonction a minimiser,
                   # On met les bons parametres
                   -get_likelihood(beta_vec = b, X = design_matrix,
                                   y = y_vector, return_log = TRUE)
                   })  %>%  # Renvoie une liste
  {.$value * (-1)} %>% # dont on extrait la valeur du minimum * -1
  exp() # on se débarasse du logarithme
```

La valeur optimale est donc d'à peu près 0.015. On est ainsi 66 fois plus efficace (en terme d'acceptation) que la borne $\tilde{M} = 1$.

On peut ainsi coder l'algorithme d'acceptation rejet. Dans cette fonction, en prévision de la suite, on conservera tous les candidats rejetés.

```
get_sample_accept_reject <- function(X,
                                     y,
                                     M_bound){
  stop_condition <- FALSE # Stopping condition
  output_matrix <- NULL # Output matrix, will contain all candidates, rejected
  # and accepted
  while(!stop_condition){
    candidate <- rnorm(n = 4, mean = 0, sd = 2) # sample from N(0, 4I)
    likelihood <- get_likelihood(candidate, X, y) # Compute likelihood
    u <- runif(1, 0, 1)
    stop_condition <- (u < (likelihood / M_bound)) # Updating stopping condition
    output_matrix <- rbind(output_matrix, candidate) # Keeping candidates
  }
  colnames(output_matrix) <- paste0("beta_", 0:3) # On nomme les colonnes
  # de la matrice
  as_tibble(output_matrix) %>% # Transformation en tibble
    mutate(accepted = c(rep(FALSE, nrow(.) - 1), TRUE)) %>% # Ajout d'une colonne
    # accepted
    return()
}
```

On peut ainsi obtenir les échantillons obtenus par de l'acceptation rejet. On le fait pour un échantillon de taille 10000.

```r
n_samples <- 1e3
# Peut prendre un peu de temps
set.seed(123) # Pour la reproductibilité
acceptance_reject_candidates <- rerun(n_samples,
                                      get_sample_accept_reject(X = design_matrix,
                                                               y = y_vector,
                                                               M = M_tilde)) %>%
  bind_rows(.id = "Replicate")
dim(acceptance_reject_candidates) # Dimension du résultat
```

```
[1] 386425      6
```

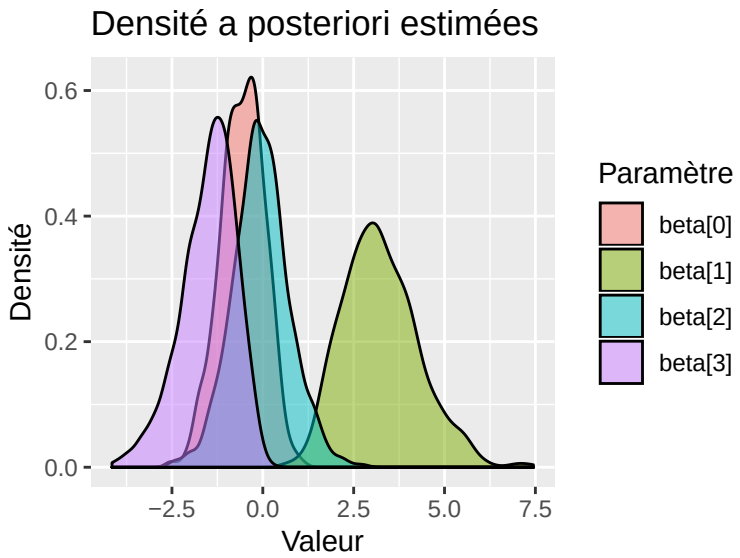
On peut remarquer qu'il y a plus de 350 refusés pour un accepté. On peut espérer utiliser ces candidats pour améliorer certaines de nos estimations.

Si l'on ne considère que les acceptés, on peut obtenir les quantités habituelles.

Ici l'intervalle de confiance est fait composante par composante. Techniquement, on pourrait déterminer un ellipse de confiance pour chaque couple de paramètres.

```r
accepted_samples <- acceptance_reject_candidates %>%
  filter(accepted) %>% # On ne garde que les acceptes
  mutate(Replicate = as.numeric(Replicate)) %>% # On crée modifie Replicate
  select(-accepted) %>% # On vire la colonne accepted
  gather(-Replicate, key = "parameter", value = "value") # On passe en format long
  # pour les graphiques

# Plot posterior densities
ggplot(accepted_samples) +
  aes(x = value) +
  geom_density(aes(fill = parameter), alpha = 0.5) +
  labs(x = "Valeur", y = "Densité", fill = "Paramètre",
       title = "Densité a posteriori estimées")
```




```r
# Intervalle de confiance a posteriori pour la moyenne
accepted_samples %>%
  group_by(parameter) %>%
  summarise(ic_95_inf = mean(value) - 1.96 * sqrt(var(value) / n()),
```

```
            posterior_mean = mean(value),
            ic_95_sup = mean(value) + 1.96 * sqrt(var(value) / n()))
```

```
# A tibble: 4 x 4
  parameter ic_95_inf posterior_mean ic_95_sup
  <chr>         <dbl>          <dbl>     <dbl>
1 beta[0]     -0.597         -0.560   -0.523
2 beta[1]      3.18           3.24     3.30
3 beta[2]     -0.0941        -0.0459   0.00219
4 beta[3]     -1.55          -1.50    -1.45
```

9. À partir de cette méthode, et en utilisant les questions 3 et 4, donner une estimation ainsi qu'un intervalle de confiance pour à 95% de la quantité

$$\int_{\mathbb{R}^4} \pi(\theta) L(y_{1:n}\theta) \mathrm{d}\theta$$

Il suffit simplement d'implémenter les formules vues plus haut. À partir du tableau des candidats, on peut extraire les temps d'attente observés. On note `I_hat` la quantité recherchée

```
nb_essais <- acceptance_reject_candidates %>%
  group_by(Replicate) %>%
  summarise(nb_essai = n()) %>%
  pull(nb_essai)
tau_hat <- mean(nb_essais)
I_hat <- M_tilde / tau_hat # Estimation
S_square <- var(nb_essais)
# Intervalle de confiance
(IC_95_I_hat <- I_hat + c(-1, 1) * qnorm(0.975) * M_tilde / tau_hat^2 * sqrt(S_square / n_samples))
```

```
[1] 3.688814e-05 4.150727e-05
```

10. Afin d'estimer $\mathbb{E}[\theta|y_{1:n}]$, implémenter l'estimateur $\hat{J}_N$ de la question 6 (avec le même algorithme d'acceptation rejet que pour la question 8). Donnez un intervalle de confiance asymptotique pour l'estimation obtenue.

À M connu, la réponse est un application directe de la section précédente. Malheureusement, ici, on ne connait que $\tilde{M}$. Par l'exercice précédent (pour une densité cible f connue seulement au travers de $\tilde{f}$), on sait que pour chaque $\tilde{M}$ valable, correspond un $M = \frac{\tilde{M}}{I(\tilde{f})}$. Dans ce cas, effectuer l'acceptation rejet avec f, g et M est équivalent (en terme de probabilité d'acceptation) qu'effectuer l'acceptation rejet avec $\tilde{f}$, g. Il suffit donc de remplacer M par $\tilde{M}/I(\tilde{f})$ dans la formule et tout doit fonctionner.

Malheureusement $I(\tilde{f})$ est inconnue. En remplaçant $\tilde{M}/I(\tilde{f})$ par son estimateur Monte Carlo naturel qui est $\hat{M} = \frac{\tilde{M}}{\tilde{I}_n(\tilde{f})} = \frac{N}{n}$, on obtient l'estimateur

$$\tilde{J}_n = \frac{1}{N}\left(\sum_{i=1}^{n} \varphi(X_i) + (\frac{N}{n} - 1) \sum_{j=1}^{N-n} \frac{\tilde{f}(Z_j)}{\tilde{M} g(Z_j) - \tilde{f}(Z_j)} \varphi(Z_j)\right)$$

Cet estimateur est désormais biaisé, mais consistant. Montrer cette consistance n'est pas trivial et cela dépasse le cadre de l'exercice (et de ce cours), donc on l'admettra. De même, la variance asymptotique de cet estimateur ne va pouvoir s'exprimer facilement ici.

Je n'avais pas vu ce détail en rédigeant le sujet, je ne noterai donc pas cette question. Je tiens à 'excuser

auprès de celles et ceux qui se seraient arracher les cheveux là dessus. Celles et ceux ayant eu des idées pertinentes auront un bonus (notamment, de remplacer M par son estimateur). Les étudiant(e)s intéressés pour approfondir pourront lire l'article de référence: *Post-Processing Accept-Reject Samples: Recycling and Rescaling* de G. Casella et C. Robert (1998).

Toujours est il que cet estimateur est implémentable facielement.

```
# Fonctions g, f, et de calcul des poids d'importances
get_g <- function(beta_vec){
  prod(dnorm(beta_vec, 0, 2))
}
get_f <- function(beta_vec){
  get_g(beta_vec) * get_likelihood(beta_vec, design_matrix, y_vector)
}
get_is_weight <- function(beta_vec, M_estimate, M_tilde,
                          accepted = FALSE){
  if(!accepted){
    f_beta <- get_f(beta_vec)
    return((M_estimate - 1) * f_beta /
             (M_tilde * get_g(beta_vec) - f_beta))
  }
  else
    return(1)
}
```

```
# Ici, un peu de préparation de donnees
# On cree un tableau ayant en colonne les argiments nécessaires
arguments_tibble <-
  acceptance_reject_candidates %>%
  select(-Replicate) %>%
  rowid_to_column(var = "index") %>%
  nest(beta_vec = c(beta_0, beta_1, beta_2, beta_3)) %>%
  mutate(beta_vec = map(beta_vec, function(x) as.numeric(x))) %>%
  select(-index)
# Maintenant, on cree le bon tableau
estimates_recyling_ar <- acceptance_reject_candidates %>%
  mutate(is_weights = pmap_dbl(.l = arguments_tibble, .f = get_is_weight,
                               M_estimate = tau_hat, M_tilde = M_tilde)) %>%
  mutate(iteration = 1:n()) %>%
  mutate_at(.vars = vars(starts_with("beta_")),
            .funs = function(colonne) cumsum(colonne * .$is_weights) / .$iteration) %>%
  mutate(methode = "AR + Recyclage") %>%
  select(-Replicate, -accepted, -is_weights) %>%
  slice(seq(1, n(), by = 100)) %>% # Only kepp one point in 100
  gather(-iteration, -methode, key = "Parametre", value = "value")
```

```
estimates_classic_ar <- acceptance_reject_candidates %>%
  filter(accepted) %>%
  mutate(iteration = 1:n()) %>%
  mutate_at(.vars = vars(starts_with("beta_")),
            .funs = function(colonne) cumsum(colonne) / .$iteration) %>%
  mutate(methode = "AR classique") %>%
  select(-Replicate, -accepted) %>%
  gather(-iteration, -methode, key = "Parametre", value = "value")
# On crée une colonne de poids, valant 1 si l'échantillon a été acceptpé,
```

```
# donnée par get_is_weight sinon
```

11. Comparez les deux estimateurs et commentez.

> Pour un $\hat{M}$ fixé, on trace les réalisation des estimateurs au cours des itérations. On constate que la variance de l'estimateur avec recyclage est bien plus forte (et on ne prend pas en compte ici la variance de $\hat{M}$)! On voit que cette variance est bien plus forte même si le nombre d'échantillons total est beaucoup plus grand.
>
> On peut ainsi voir ici qu'inclure tous les échantillons peut venir au prix d'une variance plus forte. L'idée du recyclage, séduisante en théorie, n'est pas toujours bonne en pratique.

```
bind_rows(estimates_recyling_ar,
          estimates_classic_ar) %>%
  ggplot(aes(x = iteration, y = value)) +
  geom_path(aes(color = Parametre)) +
  facet_wrap(.~methode, scales = "free_x") +
  theme(axis.text = element_text(angle = 45)) +
  labs(x = "Iteration", y = "Valeur estimée",
       title = "Estimation de l'espérance a posteriori")
```