Projet Monte Carlo

Laure Michaud - Avigaïl Zerrad

08/01/2020

Exercice 1

Partie 1: Simulations de variables aléatoires

Merci de bien vérifier que la library(reliaR) est bien installée afin d'utiliser le package gumbel. Si dans la question 1b, la densité estimée (courbe jaune) est représentée par une droite sur l'axe des abscisses ou que la première bissectrce et le qqplot ne sont pas alignés, cela veut dire que le mauvais package gumbel a été téléchargé.

```
library(reliaR)
library(plot3D)
library(viridisLite)
library(microbenchmark)
```

Paramètres de l'énoncé :

```
n <- 100
mu <- 1
beta <- 2
```

Question 1a Pour simuler suivant la densité f, nous allons utiliser la méthode de la fonction inverse. Tout d'abord, la fonction de répartition de la loi Gumbel de paramètres μ=1 et β=2 est donnée par:

$$F(x) = \exp^{-\exp(\frac{-(x-\mu)}{\beta})}$$

Si $y \in [0,1]$ on a que F(x) = y $\iff$ x = -$\beta \ln(-\ln(y)) + \mu$

Pour U$\sim \mathcal{U}[0,1]$, $F^{\leftarrow}(U) = -\beta \ln(-\ln(U)) + \mu$

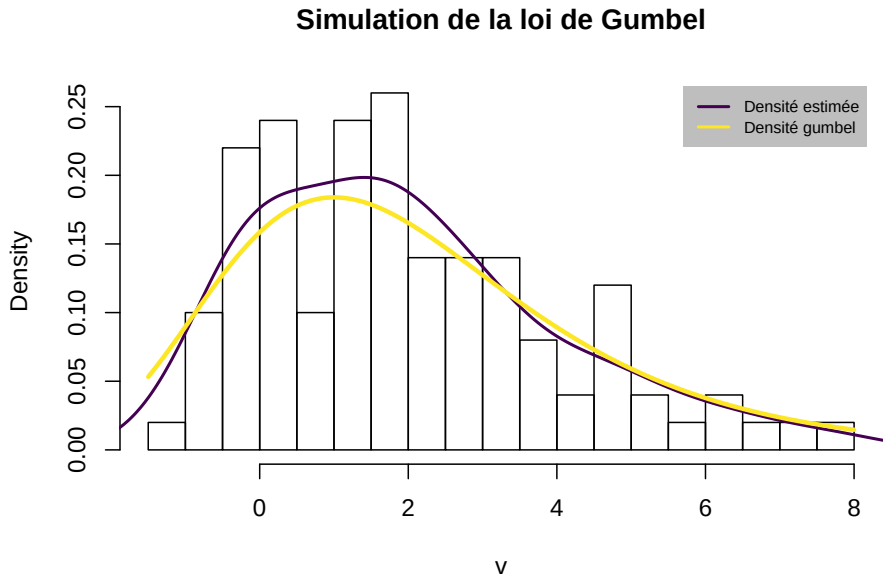
Si $y \notin [0,1]$, l'équation $F(X) = y$ n'a pas de solutions. On obtient donc le code suivant:

```
simu <- function(n){
u <- runif(n)
v <- -beta*log(-log(u))+mu
return(v)}
```

```
resultat100 <- simu(n)
resultat10 <- simu(10)
```

Question 1b Nous allons utiliser deux outils de comparaison, afin de valider notre méthode de simulation Représentons tout d'abord l'histgramme de notre échantillon.

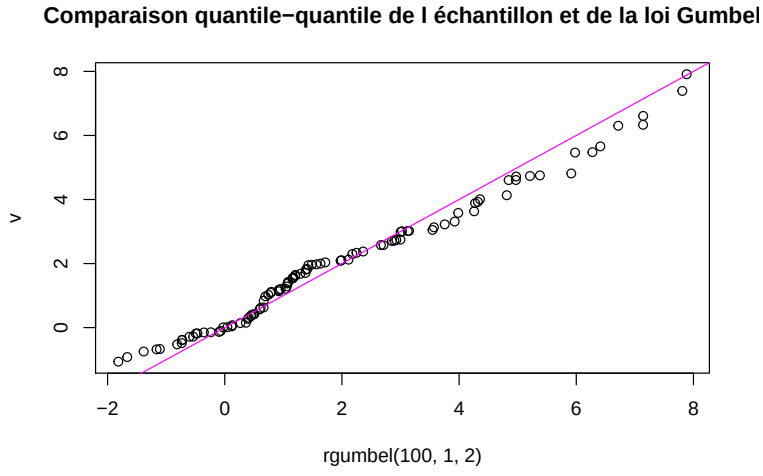
```r
v <- simu(n)
c.pal <- viridis(2)
hist(v,freq=FALSE,breaks=20,main='Simulation de la loi de Gumbel')
lines(density(v),lwd=2,col=c.pal[1])
x <- seq(-5,15)
curve(dgumbel(x,1,2),col=c.pal[2],add=T,lwd=3)
legend("topright",c("Densité estimée","Densité gumbel"),
       col=c.pal,lwd=2,box.lty=0, bg="grey",inset=.02,cex=0.7)
```



On observe que la densité estimée et la densité de la loi Gumbel coïncident.

Comme deuxième méthode, utilisons une comparaison quantile-quantile

```r
qqplot(rgumbel(100,1,2),v, main='Comparaison quantile-quantile de l échantillon et de la loi Gumbel')
abline(0,1,col="magenta")
```



De même, les quantiles de notre échantillon coïcident avec ceux de la loi Gumbel (alignement des points sur la première bissectrice). Ces outils graphiques nous permettent de valider notre méthode de simulation.

Question 2a soit $x \leq y$ Par la formule des probabilités totales, on obtient:

$\mathbb{P}(max(X_1, ..., X_n)) \leq y)$
$= \mathbb{P}(\{max(X_1, ..., X_n)) \leq y\} \bigcap \{min(X_1, ..., X_n)) \leq x\}) + \mathbb{P}(\{max(X_1, ..., X_n)) \leq y\} \bigcap \{min(X_1, ..., X_n)) > x\})$

Donc, $\mathbb{F}_{X_{(1)},X_{(2)}}(x, y) = \mathbb{P}(X_n \leq y)^n - \mathbb{P}(\{X_n \leq y\} \bigcap \{X_1 > x\}))^n = (\mathbb{F}(y))^n - (\mathbb{F}(y) - \mathbb{F}(x))^n$

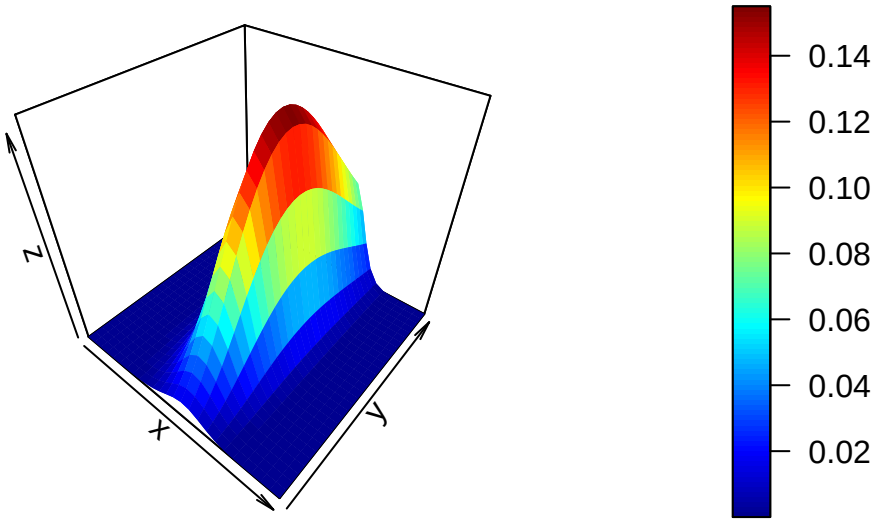
Ainsi, $\frac{\partial \mathbb{F}_{X_{(1)},X_{(2)}}(x,y)}{\partial x} = nf(x)(\mathbb{F}(y) - \mathbb{F}(x))^{n-1}$ et on a: $\frac{\partial \mathbb{F}_{X_{(1)},X_{(2)}}(x,y)}{\partial x \partial y} = n(n-1)f(x)f(y)(\mathbb{F}(y) - \mathbb{F}(x))^{n-2}$

Au final, on obtient bien: $f_{1,n}(x, y) = n(n-1)f(x)f(y)(\mathbb{F}(y) - \mathbb{F}(x))^{n-2}1_{x \leq y}$.

Traçons une représentation de la densité $f_{1,n}$:

```
f1n<-function(x,y){
  n*(n-1)*(pgumbel(y,mu,beta)-pgumbel(x,mu,beta))^(n-2)*dgumbel(y,mu,beta)*dgumbel(x,mu,beta)*(x<=y)}

x<-seq(-5,0,0.2)
y<-seq(7,13,0.2)
z<-with(mesh(x,y),f1n(x,y))
persp3D(z=z,x=x,y=y,title='représentation de la densité f_{1,n}')
```



Question 2b Pour simuler la densité $f_{1,n}$, utilisons la méthode de rejet. Remarquons tout d'abord que $f_{1,n}(x, y) \leq n(n-1)f(x)f(y)$ Posons $M = n(n-1)$. Soit $X = (X_1, X_2)$ une variable aléatoire de densité $f_{1,n}$ de $\mathbb{R}^2$. Soient $(U_n)_{n \geq 1}$ v.a iid $\mathbb{U}[0, 1]$ indépendantes de $(Y_n)_{n \geq 1} = (Y_{1,n}, Y_{2,n})$, va iid de loi de densité $g(x, y) = f(x)f(y)$.

Soit $T = \inf \left\{n \geq 1, U_n \leq (\mathbb{F}(y) - \mathbb{F}(x))^{n-2} 1_{Y_{1,n} \leq Y_{2,n}}\right\}$,on a alors $Y_T \sim f_{1,n}$ On simule donc $Y_{1,n} \sim f$ $Y_{2,n} \sim f$.

On remarque que la probabilité d'acceptation est $p = \frac{1}{M} = \frac{1}{n(n-1)}$. Pour obtenir un échantillon de taille m, il faudra en moyenne $\frac{m}{p} = Mm = n(n-1)M$ tirages suivant g. Réalisons tout d'abord un algorithme du rejet basique:

```
rejetbasique<-function(m,n){
  s <- c()
for (i in 1:n){
  x <- rgumbel(2,mu,beta)
  u <- runif(1,0,1)
  while (u > (pgumbel(x[2],mu,beta) - pgumbel(x[1],mu,beta))^(n-2)*(x[2] - x[1] > 0)){
```

```
    x <- rgumbel(2,mu,beta)
    u <- runif(1,0,1) }
  s <- c(s,x)}
s <- matrix(s,nrow=2)

return(s)}
```

```
rejetbasique(100,n) # pour n=100 (le n de l'énoncé)
rejetbasique(100,10)  # pour n=10
```

Cet algorithme prend beaucoup de temps à s'éxécuter du fait de la faible probabilité d'acceptation. Regardons maintenant une version vectorisée de cet algorithme:

```
rejet2 <- function(k,n){
  x1 <- c()
  x2 <- c()
  m <- k # nbr qu'il reste à simuler

  while(m>0){

    y1 <- simu(n*(n-1)*m)
    y2 <- simu(n*(n-1)*m)
    u <- runif(n*(n-1)*m)
    indice<-(y1<=y2)
    v <- (pgumbel(y2,mu,beta)-pgumbel(y1,mu,beta))**(n-2)
    v <- v*indice
    w <- (u<=v)
    x1 <- append(x1,y1[which(w)])
    x2 <- append(x2,y2[which(w)])
    m <- n-length(x1)
  }
  return(matrix(c(x1,x2),nrow=2,byrow=TRUE))
}
```

Partie 2: Estimation de l'étendue par la méthode de Monte Carlo classique

Question 1a Définissons tout d'abord un estimateur de Monte Carlo classique. $\widehat{\delta_m} = \frac{1}{m}\sum_{i=1}^{m}\Delta_i$ où $\Delta_i = X^i_{(n)} - X^i_{(1)}$ avec $X^i \sim f$ On construit un intervalle de confiance au niveau 95% de $\widehat{\delta_m}$. On obtient alors: $IC_{1-\alpha} = [\widehat{\delta_m} - \frac{q\sigma}{\sqrt{m}}, \widehat{\delta_m} + \frac{q\sigma}{\sqrt{m}}]$ où q est le quantile $(1-\frac{\alpha}{2})$de la loi normale $\mathcal{N}(0,1)$.

Par conséquent, la précision est donnée par $\epsilon = \frac{q\sigma}{\sqrt{m}} = 0.01$ On doit donc avoir $m = (\frac{q\sigma}{0.01})^2$. σ étant inconnue, nous avons deux stratégies possibles: utiliser la fonction MC.estim.evol dans laquelle nous avons rajouté une sortie précision qui nous renvoie: $\frac{q\sigma}{\sqrt{m}}$ et sélectionner la valeur minimale de m pour laquelle la précision est atteinte. Cependant, cette méthode prend énormément de temps à s'éxécuter.

```
MC.estim.evol<-function(y,level=0.95){
  n=length(y)
  delta<-cumsum(y)/(1:n)
  s2<-(cumsum(y*y)-(1:n)*(delta)^2)/(0:(n-1))
  s2<-c(0,s2[-1])/(1:n)
  b_IC<-qnorm(0.5*(1+level))*sqrt(s2)
```

```
  return(list(value=delta,var=s2,IC=c(delta-b_IC,delta+b_IC),precision=b_IC))
}
```

```
MC.estim<-function(y,level=0.95){
  m<-mean(y)
  s2<-(1/length(y))*var(y)
  edelta<-qnorm(0.5*(1+level))*sqrt(s2)
  ICinf<-m-edelta
  ICsup<-m+edelta
  return(list(delta=m,var=s2,binf=ICinf,bsup=ICsup))
}
```

Afin d'utiliser la densité f, nous créons la fonction estim1 qui utilise la fonction simu pour simuler Δ_i.

```
estim1<-function(m,n){
z<-numeric(m)
for  (i in 1:m){
  w<-simu(n)
  z[i]<-max(w)-min(w)}
return(z)}
```

En simulant plusieurs échantillons Δ_i, on observe que leur variance est de l'ordre de 8. On estime donc notre σ^2 inconnu par 8. On obtient alors le code suivant

```
msimu1 <- 8*(qnorm(1-0.05/2)/0.01)^2
y <- estim1(msimu1,n)
(resultat <- MC.estim(y))
```

```
## $delta
## [1] 13.6009
##
## $var
## [1] 2.217047e-05
##
## $binf
## [1] 13.59167
##
## $bsup
## [1] 13.61012
```

```
(variance <- resultat$var)
```

```
## [1] 2.217047e-05
```

Notons que cette méthode n'est pas optimale car on utilise une estimation de la variance obtenue sur un échantillon. Une autre méthode considérerait à majorer σ, afin d'obtenir une approximation de m. Cependant, en la réalisant nous aboutissons à des majorations de la variance beaucoup trop grossières. Le code ci-dessous utilise la fonction MC.estim.evol mais comme évoqué ci-dessus son temps d'éxécution est très long, c'est pourquoi à l'éxécution, il ne tourne pas ('Rsession aborted'). Son principe était de simuler un échantillon suffisamment grand à l'aide de la fonction estim1 puis de lui appliquer la fonction MC.estim.evol pour récupérer la précision. Ensuite, on sélectionne les indices du vecteur renvoyé par MC.estim.evol dont la précision est inférieure ou égale à 0.01. Puis on retourne le plus petit indice pour lequel la précision est réalisée.

```r
# NE PAS EXECUTER (SESSION R ABORTED)
n <- 600000
v <- estim1(n)
w <- MC.estim.evol(v) # on stocke le résultat dans un vecteur w.

resultat <- w$precision    # on extraie la précision.

indice <- (resultat<0.01)   # on sélectionne les termes de précision inférieure à 0.01.

msimu <- min((which(indice))[2]) # On sélectionne le plus petit indice pour laquelle
#la précision est réalisée. On enlève le 1 er terme de MC evol qui vaut toujours 1.

w[msimu]  # On obtient alors la valeur de notre estimateur.
```

Question 1b Utilisons maintenant la fonction $f_{1,n}$. On réapplique la même méthode que précedemment, en utilisant à la place de estim1 la fonction rejet de la question 2b. Nous obtenons le même problème. En effet, cela peut s'expliquer par la très faible probabilité d'acceptation de notre méthode de rejet. Pour pâlier à ce problème, il faudrait appliquer une méthode de rejet avec un M plus petit afin d'augmenter notre probabilité d'acceptation.

NE PAS EXECUTER (SESSION R ABORTED)

```r
u <- rejet2(3000,n)
v2 <- u[2,]-u[1,]
w2 <- MC.estim.evol(v2)
resultat2 <-w2$precision
indice2 <- (resultat2<0.01)
msimu2 <- min((which(indice2))[2])
w2[msimu2]
```

Efficacité relative de ces deux méthodes:

```r
MC_f <- function(n,m){
  y <- estim1(m,n)
  x <- MC.estim(y)
  return(x$delta)
}

MC_rejet <- function(n,m){
  u <- rejet2(m,n)
  y <- u[2,]-u[1,]
  x <- MC.estim(y)
  return(x$delta)
}
```

```r
microbenchmark(MC_f(100, 10), MC_rejet(100,10))
```

```
## Unit: microseconds
##               expr        min         lq        mean     median
##      MC_f(100, 10)    137.345    163.558    247.4919    218.608
##  MC_rejet(100, 10) 366845.120 435672.939 475398.4888 467500.569
##           uq        max neval
##     270.7725   2767.421   100
##  504660.8095 863031.064   100
```

Interprétation: La méthode du rejet prend plus de temps. Cela s'explique par la faible probabilité d'acceptation pour la méthode du rejet de l'ordre de $(1/n^2)$. Il faudrait aussi comparer les variances des 2 méthodes.

Question 2a Nous allons utiliser la fonction score afin de construire une estimation de δ par la méthode de la variable de contrôle.En effet pour un h_0 bien choisie (d'espérance facile à calculer et tel que $\text{Var}(h_0(X_k))$ finie), on aura pour tout réel b positif, étant donnée une suite $(Z_n)_{n\geq 1}$ de variables aléatoires iid suivant la loi Δ, on obtient l'estimateur suivant: $\widehat{\delta_n} = \frac{1}{n}\sum_{k=1}^{n}[(h(X_k) - b(h_0(X_k) - m)]$ Tout d'abord, la log-vraisemblance est donnée par: $log(f(x_1, ...x_n) = -n\log(\beta) - \sum_{i=1}^{n}(\frac{x_i-\mu}{\beta}) - (\sum_{i=1}^{n}\exp(-\frac{x_i-\mu}{\beta}))$

Calculons la dérivée de la log-vraisemblance par rapport au paramètre β:

$\frac{\partial \log f(x_1,...,x_n)}{\partial \beta} = -\frac{n}{\beta} - \sum_{i=1}^{n}[\frac{x_i-\mu}{\beta^2}\exp^{(}-\frac{x_i-\mu}{\beta})] + \sum_{i=1}^{n}\frac{x_i-\mu}{\beta^2}$ On sait que l'espérance de la fonction score vaut 0 (donc m = 0). Nous allons utiliser pour h_0 la première composante de la fonction score et en déterminant le b optimal par période de chauffe: $\widehat{b_l^*} = \frac{\sum_{k=1}^{l}[(h_0(X_k)-m)(h(X_k)-\overline{h}_l)]}{\sum_{k=1}^{l}[(h_0(X_k)-m)^2]}$, nous obtenons alors l'estimateur $\widehat{\delta_{n-l}}(\widehat{b_l^*})$

```
h1 <- function(y){
  n <- length(y)
  return(-n/beta-sum((exp(-(y-mu)/beta))*(y-mu)/(beta*beta))+sum((y-mu)/(beta*beta)))}
```

```
# Estimons b par période de chauffe en prenant h0 = dér partielle % à beta.

m <- 100
n <- 100  # le n demandé pour les implémentations
l <- 20 # nous utilisons 20% des données simulées

controle <- function(m,n,l){
  h0 <- c()
  h <- c()
  for  (i in 1:m){
    w <- simu(n)
    h0 <- c(h0,h1(w))
    h<-c(h,max(w)-min(w))} #construction de h et h0

  hl <- rep(MC.estim(h[1:l])$delta,l)

  numerateur <- sum(((h0[1:l]))*(h[1:l]-hl))
  denominateur <- sum((h0[1:l])^2)
  bstar <- numerateur/denominateur #on a utilisé les l premières valeurs

  y <- h[-(1:l)]-bstar*((h0[-(1:l)])) #on utilise le reste pour estimer delta
  return(y)}


z <- controle(m,n,l)
MC.estim(z)
```

```
## $delta
## [1] 13.48416
##
## $var
## [1] 0.07589994
##
```

```
## $binf
## [1] 12.94419
##
## $bsup
## [1] 14.02413
```

Question 2b Pour comparer les variances entre la méthode de Monte-Carlo classique et la méthode de la variable de contrôle, on doit prendre le même echantillon.

```
compareVar<-function(n,m){
  z <- c()
  h0 <- c()
  h <- c()
  for  (i in 1:m){
    w <- simu(n)
    h0 <- c(h0,h1(w))
    h<-c(h,max(w)-min(w))}

  hl <- rep(MC.estim(h[1:l])$delta,l)
  numerateur <- sum(((h0[1:l]))*(h[1:l]-hl))
  denominateur <- sum((h0[1:l])^2)
  bstar <- numerateur/denominateur

  y <- h[-(1:l)]-bstar*((h0[-(1:l)]))

  return(list(var_MCclassique=MC.estim(h)$var,var_Controle=MC.estim(y)$var))}
```

```
compareVar(100,1000)
```

On remarque que les variances pour les deux méthodes sont assez proches, avec une variance tout de même légèrement plus faible pour la méthode de contrôle.

La méthode de la variable de contrôle est donc plus efficace si le temps de calcul est plus faible que pour la méthode classique. Estimons les temps de calcul MOYENS de chaque méthode :

```
estim<-function(n,m){     #en pratique, on prend n = 100
  z<-c()
  for  (i in 1:m){
    w<-simu(n)
    z<-c(z,max(w)-min(w))}
  return(z)}

MC_classique <- function(n,m){
  y <- estim(n,m)
  x <- MC.estim(y)
  return(x$delta)
}
MC_controle <- function(n,m,l){
  y <- controle(n,m,l)
  x <- MC.estim(y)
  return(x$delta)
}
```

```
microbenchmark(MC_classique(100, 100), MC_controle(100,100,20))
```

```
## Unit: milliseconds
##                       expr      min       lq     mean   median       uq
##     MC_classique(100, 100) 1.122826 1.678341 2.472558 1.954098 2.407497
##  MC_controle(100, 100, 20) 1.925161 2.768613 3.692226 3.217915 3.885690
##       max neval
##  28.83778   100
##  20.00302   100
```

```
microbenchmark(MC_classique(100, 1000), MC_controle(100,1000,20))
```

```
## Unit: milliseconds
##                        expr      min       lq     mean   median       uq
##     MC_classique(100, 1000) 13.03523 17.23716 20.23966 18.70439 20.64328
##  MC_controle(100, 1000, 20) 11.13965 13.25209 15.93592 14.68221 15.82541
##       max neval
##  41.83979   100
##  33.32617   100
```

La méthode de la variable de contrôle réduit le temps de calcul pour m grand. (en effet pour m = 100, le temps de calul est plus important pour MC_Controle).

Rq : à partir de m = 500 (à peu près, car il s'agit de moyennes de temps), la méthode du contrôle est bien plus rapide que MC classique.

On en déduit que l'efficacité relative (cf formule) est plus grande que 1 car C (MC_classique) > C1 (MC_controle), donc C/C1 >1 (à variances ~ égales).

d'où: pour m grand, la méthode de la variable de contrôle est plus efficace.

Partie 3: Estimation d'un évènement rare

Paramètres de l'énoncé:

```
t <- 8
n <- 1000
```

Question 1 $V_1, ...V_n$ iid $\mathcal{E}(2)$ avec $V_{(n)} = max(V_1, ...V_n)$ On remarque que $p = \mathbb{P}(V_{(n)} \geq t) = \mathbb{E}[1_{V_{(n)} \geq t}]$. Utilisons un estimateur de Monte Carlo classique: $\widehat{\delta_m} = \frac{1}{m} \sum_{i=1}^{m} 1_{V^i_{(n)} \geq t}$. On obtient donc l'intervalle de confiance classique au niveau 95%:

```
m <- 1000
simu3 <- function(m){
  w <- c()
  for (i in 1:m){
  v <- max(rexp(n,2))
  w <- c(w,v)
  }
  return(w)}

v <- (simu3(m)>=8)
MC.estim(v)
```

```
## $delta
## [1] 0
##
## $var
## [1] 0
##
## $binf
## [1] 0
##
## $bsup
## [1] 0
```

La simulation "naïve" est inefficace puisqu'il s'agit d'un évènement rare. Une solution consite donc à forcer la réalisation de cet évènement comme le suggère la deuxième question.

Question 2a $\mathbb{P}(V_{(n)} - 0.5log(n) \leq k) = \mathbb{P}(V_{(n)} \leq k + 0.5log(n)) = \mathbb{P}(V_1 \leq k + 0.5log(n))^n = (1 - \exp^{-2(k+0.5\log(n))})^n = \exp^{n\ln(1-\exp^{-2(k+0.5\log(n))})} = \exp^{n\ln(1-\frac{e^{-2k}}{n})}$ or $\ln(1 - \frac{e^{-2k}}{n}) \sim -\exp^{-2k}$ qui est une constante. On a donc $\lim_{n\to+\infty} \mathbb{P}(V_{(n)} - 0.5log(n) \leq k) = \exp^{-e^{-2k}}$. On a donc une convergence en loi vers une loi Gumbel de paramètres $\mu = 0$ et $\beta = 0.5$.

Question 2b Utilisons une méthode d'échantillonnage préférentiel pour estimer p. Par la question précédente, on remarque que pour n assez grand, on a une convergence en loi vers une Gumbel. Cela nous donne donc une idée pour le choix de notre densité instrumentale. On va donc utiliser une densité d'une loi Gumbel. Nous avons choisi la loi gumbel (3,4). En effet, grâce à une translation, nous forçons la réalisation de l'évènement $V_{(n)} \geq t$ Adoptons les notations suivantes: f la densité de $max(V_1, ..., V_n)$ avec $V_i \sim \mathcal{E}(2)$ $X = V_n$; $h = 1_{(.\geq t)}$; Z variable aléatoire suivant la densité g; g la densité de la loi Gumbel(3,4). On remarque que supp(f)$\subset$ supp(g) et le rapport $\frac{f}{g}$ est borné. On obtient donc bien $\mathbb{E}_f[h(X)] = \mathbb{E}_g[\frac{h(Z)f(Z)}{g(Z)}]$.

Etant donnée $(Z_n)_{n\geq 1}$ suite de variables aléatoires iid suivant la densité g, on définit l'estimateur d'échantillonnage préférentiel par :

$\widehat{\delta_m} = \frac{1}{m}\sum_{i=1}^{m} \frac{f(Z_k)}{g(Z_k)} h(Z_k)$

```
# bloc génération
z <- rgumbel(m,3,4)
w <- dexp(z,2)*n*(pexp(z,2))^(n-2)/dgumbel(z,3,4)
wfinal <- w*(z>=t)

# bloc simulation
MC.estim(wfinal)
```

```
## $delta
## [1] 0.0001059239
##
## $var
## [1] 1.901898e-10
##
## $binf
## [1] 7.889414e-05
##
## $bsup
## [1] 0.0001329536
```

Question 3: efficacité relative des deux méthodes Calculons les coûts de chaque méthode:

```r
MC <- function(m){
  y <- (simu3(m)>=8)
  x <- MC.estim(y)
  return(x$delta)
}

MC_pref <- function(m){
  z <- rgumbel(m,3,4)
  w <- dexp(z,2)*n*(pexp(z,2))^(n-2)/dgumbel(z,3,4)
  y <- w*(z>=t)
  x <- MC.estim(y)
  return(x$delta)
}
```

```r
microbenchmark(MC(1000), MC_pref(1000))
```

```
## Unit: microseconds
##           expr       min         lq       mean     median         uq
##       MC(1000) 76609.717 82732.0505 87015.7530 85751.0075 90149.0435
##  MC_pref(1000)   262.503   324.0325   658.5938   421.4355   476.6075
##        max neval
##  146699.97   100
##   13877.95   100
```

On remarque que les variances pour les deux méthodes sont assez proches, avec une variance tout de même légèrement plus faible pour la méthode de Mote Carlo classique.

Le temps de calcul est environ 100 fois moins important pour la méthode d'échantillonnage préférentiel avec m=1000 simulations. Cette méthode est donc beaucoup plus efficace.

Question 4a On utilise le fait que si $U \sim \mathbb{U}[0,1]$ alors $1-U \sim \mathbb{U}[0,1]$ On obtient à partir de notre estimateur d'échantillonnage préférentiel, l'estimateur suivant: $\widehat{\delta_m} = \frac{1}{2m}\sum_{i=1}^{m} \frac{f(Z_k)}{g(Z_k)}h(Z_k) + \frac{f(A(Z_k))}{g(A(Z_k))}h(A(Z_k))$ où $Z_k = F^{\leftarrow}(U_k)$ et $A(Z_k) = F^{\leftarrow}(1-U_k)$. Pour ce faire, on utilise la fonction simuanti qui permet de simuler les $Z_k = F^{\leftarrow}(U_k)$ et les $F^{\leftarrow}(1-U_k)$. On utilise le même échantillon U_k.

```r
simuanti <- function(n){
  u <- runif(n)
  v1 <- -4*log(-log(u))+3
  v2 <- -4*log(-log(1-u))+3
  resultat <- c(v1,v2)

  return(resultat)}
```

```r
n1 <- 100
z <- simuanti(n1)
w1 <- dexp(z,2)*n*(pexp(z,2))^(n-2)/dgumbel(z,3,4)
wfinal1<-w1*(z>=t)

MC.estim(wfinal1)
```

```
## $delta
## [1] 8.846329e-05
##
```

```
## $var
## [1] 6.816704e-10
##
## $binf
## [1] 3.729094e-05
##
## $bsup
## [1] 0.0001396356
```

```
n2 <- 10
z2 <- simuanti(n2)
w2 <- dexp(z2,2)*n*(pexp(z2,2))^(n-2)/dgumbel(z2,3,4)
wfinal2<-w2*(z2>=t)

MC.estim(wfinal2)
```

```
## $delta
## [1] 0.0001044006
##
## $var
## [1] 3.897829e-09
##
## $binf
## [1] -1.796502e-05
##
## $bsup
## [1] 0.0002267663
```

Question 4b: efficacité relative des deux méthodes Calculons les coûts de chaque méthode:

```
MC_anti <- function(m){
  z2 <- simuanti(n2)
  w2 <- dexp(z2,2)*n*(pexp(z2,2))^(n-2)/dgumbel(z2,3,4)
  y <- w2*(z2>=t)
  x <- MC.estim(y)
  return(x$delta)
}
```

```
microbenchmark(MC_pref(1000), MC_anti(1000))
```

```
## Unit: microseconds
##           expr     min       lq     mean   median       uq       max neval
##  MC_pref(1000) 253.048 291.7790 368.6741 395.4550 409.0470   562.304   100
##  MC_anti(1000)  50.674  55.3635 216.3518  89.5315  96.4665 13536.777   100
```

On remarque que le temps de calcul est plus faible pour la méthode de la variable antithétique.

Comparons à présent les variances obtenues :

```
z <- rgumbel(m,3,4)
w <- dexp(z,2)*n*(pexp(z,2))^(n-2)/dgumbel(z,3,4)
y1 <- w*(z>=t)
```

```r
resultat1 <- MC.estim(y1)$var

z2 <- simuanti(n2)
w2 <- dexp(z2,2)*n*(pexp(z2,2))^(n-2)/dgumbel(z2,3,4)
y2 <- w2*(z2>=t)
resultat2 <- MC.estim(y2)$var

print(list(Var_pref = resultat1,Var_anti = resultat2))
```

```
## $Var_pref
## [1] 1.340361e-10
##
## $Var_anti
## [1] 4.29269e-08
```

La variance obtenue à l'aide de la méthode de la variable antithétique est donc plus faible. Cette méthode étant aussi plus rapide, on en déduit qu'elle est plus efficace que la méthode d'échantillonage préférentiel.

Exercice 2

Paramètres de l'exercice

```r
lambda1 <- 3.7 # Poisson
k <- 0.5
lambda2 <- 2  # Weibull
n <- 100
```

Question 1 On estime p par la méthode de Monte Carlo classique. On obtient alors l'estimateur suivant: $\widehat{\delta_m} = \frac{1}{m}\sum_{i=1}^{m} 1_{X_i \leq 3}$. On utilise d'abord la fonction xsimu qui nous simule un vecteur de taille n de X_i

```r
#Créons un simulateur de va X
xsimu <- function(n){
  x <- c()
  for (i in 1:n){
    s <- rpois(1,lambda1)
    if (s==0) {x<-c(x,0)}
    else {q <- rweibull(s,k,lambda2)
          x <- c(x,sum(q))}
    }
  return(x)
}
```

On sélectionne les éléments inférieurs ou égaux à 3 et on applique la fonction MC.estim au vecteur obtenu.

```r
y <- (xsimu(n)<=3)
#bloc simulation
(resultat <- MC.estim(y))
```

```
## $delta
## [1] 0.25
```

```
##
## $var
## [1] 0.001893939
##
## $binf
## [1] 0.1647035
##
## $bsup
## [1] 0.3352965
```

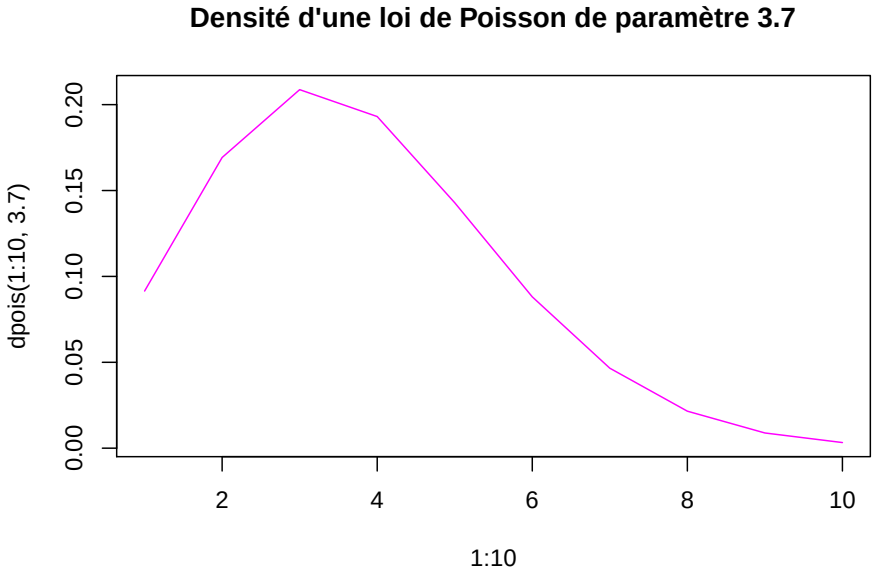
Question 2a On cherche une approximation de p par la méthode d'allocation de stratification avec allocation proportionnelle. Adoptons les notations suivantes: Z=S $\sim$ Poisson(3.7); strates $D_k = \{k\}$; allocation $p_k = \frac{n_k}{n}$ donc $n_k = p_k n$; $X|Z \in D_k$ est obtenu par la fonction xsimu. Nous sommes dans le cas d'une allocation proportionnelle nous avons donc: $q_k = p_k$ où $q_k = \frac{n_k}{n}$ et $\frac{1}{n} = \frac{p_k}{n_k}$. On obtient donc l'estimateur suivant: $\widehat{\delta_n}(n_1, ..., n_K) = \frac{1}{n}\sum_{k=1}^{K}\sum_{i=1}^{n_k} h(X_i^{(k)})$ où K est le nombre de strates. La fonction nstrate nous renvoie le nombre de tirages par strates.

```
nstrate <- function(n){

  p <- dpois(0,lambda1)
  proba <- p
  nk <- round(p*n)
  i<-1
   while(round(p*n)>0){
     p <- dpois(i,lambda1)
     proba <- c(p,proba)
     nk <- c(nk,round(p*n))
     i <- i+1}
  return(nk)}
```

On en déduit que les strates Dk sont les singletons {i} avec i allant de 0 à 9. Cela est cohérent. En effet, en visualisant la fonction de masse d'une poisson P(3.7), on remarque bien que la probabilité d'être égale 10 est très proche de 0.

```
plot(1:10,dpois(1:10,3.7),type='l',main="Densité d'une loi de Poisson de paramètre 3.7", col='magenta')
```



On simule ensuite les X_i de la strate k_0 à l'aide de la fonction xsimu_k0:

```
xsimu_k0 <- function(k0,nk0){
  x <- c()
  for (i in 1:nk0){
    if (k0==0) {x <- c(x,0)}
#si k0=0, alors X=0 car k0 représente la valeur prise pas S (= nb de précipitations)
    else {q <- rweibull(k0,k,lambda2)
    x <- c(x,sum(q))
    }
  }
  return(x)
}
```

On simule ensuite le vecteur comportant les $h(X_i^{(k)})$ auquel on applique ensuite la fonction MC.estim.

```
xsimu_strat <- function(n){
  vec_nk <- nstrate(n)
  N <- length(vec_nk)
  x <- rep(0,vec_nk[1])
  for (k0 in 1:(N-1)){
    x <- c(x,xsimu_k0(k0,vec_nk[k0+1]))
  }
  return((x<3))}


resultat3 <- xsimu_strat(n)
#bloc estimation
MC.estim(resultat3)
```

```
## $delta
## [1] 0.2673267
##
## $var
## [1] 0.001958632
##
## $binf
## [1] 0.1805857
##
## $bsup
## [1] 0.3540677
```

Question 2b Comparaison des variances:

```
compare_var2 <- function(n){
  y1 <- (xsimu(n)<3)
  resultat1 <- MC.estim(y1)$var

  y2 <- xsimu_strat(n)
  resultat2 <- MC.estim(y2)$var
  return(list(Var_classique = resultat1,Var_stratProp = resultat2))
}

compare_var2(1000)
```

```
## $Var_classique
## [1] 0.0001846486
##
## $Var_stratProp
## [1] 0.0001929139
```

On remarque que les variances sont très proches. La méthode de stratification sera donc plus efficace que MC classique si le temps de calcul est inférieur.

#comparaison des temps de calcul (coûts)

```
MC_classique2 <- function(n){
  y <- xsimu(n)
  x <- MC.estim(y)
  return(x$delta)
}

MC_stratProp <- function(n){
  y <- xsimu_strat(n)
  x <- MC.estim(y)
  return(x$delta)
}
```

```
microbenchmark(MC_classique2(100), MC_stratProp(100)) #résultat en microseconds, méthode de stratificat
```

```
## Unit: microseconds
##                expr     min       lq     mean   median      uq      max
##  MC_classique2(100) 461.008 486.9975 753.4084 802.3215 857.917 4160.947
##   MC_stratProp(100) 364.104 415.2220 775.6337 651.2475 679.014 9182.717
##  neval
##    100
##    100
```

```
microbenchmark(MC_classique2(1000), MC_stratProp(1000)) #résultat en milliseconds, méthode de stratific
```

```
## Unit: milliseconds
##                 expr      min       lq      mean   median        uq
##  MC_classique2(1000) 6.255661 8.308184 11.015803 9.479128 14.548962
##   MC_stratProp(1000) 3.164421 4.417660  5.998667 5.450066  5.933323
##       max neval
##  20.39331   100
##  17.72995   100
```

On observe que le coût de la méthode de stratification est moins important même pour des petits échantillons. Cette méthode est donc plus efficace. Ceci s'explique par le fait que la méthode de stratification permet de ne pas considérer la queue de distribution de la Poisson.

Question 3a L'estimateur de variance minimale $\widehat{\delta_n}(n_1^*, ..., n_K^*)$ est obtenu pour allocation optimale définie pour k=1,...K par $q_k^* = \frac{p_k \sigma_k}{\sum_{i=1}^K p_i \sigma_i}$. On a alors Var$[\widehat{\delta_n}(n_1^*, ..., n_K^*)]$=$\frac{1}{n}(\sum_{k=1}^K p_k \sigma_k)^2$

```
xsimu2 <- function(n){
  x <- numeric(n)
  strate <- numeric(n)

  for (i in 1:n){
    s <- rpois(1,lambda1)
    if (s==0) {x[i] <-0}
    else {q <- rweibull(s,k,lambda2)
    x[i] <-sum(q)
    strate[i] <- s
    }
  }
  return(matrix(c(x,strate),nrow=2,byrow=TRUE))
}

A <- xsimu2(1000)


xopti <- function(n){

  x <- xsimu2(n)
  strate <-as.numeric(levels(as.factor(x[2,])))
  N <- length(strate)
  vecnk <- numeric(N)


  ecarttype <- numeric(N)
  pk0 <- numeric(N)

  # i position de l'indice ; k0 est la strate

  for (i in 1:N){
    k0 <-strate[i]
    pk0[i] <- dpois(k0,lambda1)
    indicek0 <-which(x[2,]==k0)
    if (length(indicek0)>1){
    ecarttype[i] <- sd(x[1,indicek0])}
    }

  vecnk <-round(pk0*ecarttype*n/sum(pk0*ecarttype),0)

  indice <- which(vecnk!=0)
  strate <- strate[indice]  # on enlève les strates où le sd vaut 0
  vecnk <- vecnk[indice]
  pk0 <- pk0[indice]
  N <-length(strate)

  h <- numeric(N)
  for (i in 1:N){

    x <- xsimu_k0(strate[i],vecnk[i])
    y <- (x<3)
    h[i] <- sum(y)}
```

```
  resultat <- sum(h*pk0/vecnk)

  return(list(delta=resultat, var=(1/n^2)*(sum(pk0*ecarttype[1:length(pk0)])*sum(pk0*ecarttype[1:length
  }
```

```
xopti(1000)
```

```
## $delta
## [1] 0.2296715
##
## $var
## [1] 0.0001405972
```

On doit tout d'abord calculer les σ_k de chaque strate à l'aide d'une première simulation sans imposer la taille n_k de chaque strate. Ensuite, on doit enlever les écart-types nuls ou NA qui correspondent à des strates de taille 1 car q_k^* doit être non nulle (pour que $n_k^*>0$). La difficulté principale est donc la sélection de strates.

Question 3b

Comparons les variances des 3 méthodes :

```
compare_var3 <- function(n){
  y1 <- (xsimu(n)<3)
  resultat1 <- MC.estim(y1)$var

  y2 <- xsimu_strat(n)
  resultat2 <- MC.estim(y2)$var

  resultat3 <- xopti(n)$var

  return(list(Var_classique = resultat1, Var_stratProp = resultat2, Var_stratOpt = resultat3))}

compare_var3(1000)
```

```
## $Var_classique
## [1] 0.00019497
##
## $Var_stratProp
## [1] 0.0001890453
##
## $Var_stratOpt
## [1] 0.0001470076
```

Les 3 méthodes présentent des variances proches.

Comparons les temps de calcul (coûts)

```
MC_stratOpt <- function(n){
  x <- xopti(n)$delta
  return(x)
}
```

```r
microbenchmark(MC_classique2(1000), MC_stratProp(1000), MC_stratOpt(1000))
```

```
## Unit: milliseconds
##                 expr      min        lq      mean    median        uq
##  MC_classique2(1000) 6.110156  7.999689 10.995775  9.594019 13.948112
##   MC_stratProp(1000) 3.169921  4.517936  6.911739  5.609528  6.256499
##    MC_stratOpt(1000) 9.744348 12.502111 15.789759 14.862171 18.951460
##       max neval
##  20.75421   100
##  63.93458   100
##  27.75671   100
```

La méthode de stratification avec allocation proportionnelle prend environ 2 fois moins de temps à s'éxéuter que la méthode de Monte Carlo classique et 3 fois moins de temps que la méthode de stratification avec allocation optimale. On en déduit que la stratification avec allocation proportionnelle est donc plus efficace que la méthode de Monte Carlo classique qui est elle même plus efficace que la méthode de stratification avec allocation optimale.