Corrigé de l'examen NOISE de janvier 2012 (sujet B)

12 décembre 2012

1 Exercice 1

```
> mu<-1
> beta<-2
> fgumbel<-function(x,mu=1,beta=2){
+   z<-(x-mu)/beta
+   return(1/beta*exp(-z-exp(-z)))
+ }
```

1.1 Question 1

On va simuler selon la distribution de Gumbel en utilisant un algorithme d'acceptation-rejet basé sur la loi normale $N(1, s^2)$ où s est à déterminer.

Si on note $g(x)$ la densité de la Gumbel et $h(x, s)$ la densité de $N(1, s^2)$, il faut donc trouver $M(s)$ tel que

$$\forall x \in \mathbb{R}, g(x) \leq M(s)h(x, s)$$

On pose donc $M(s) = \max g(x)/h(x, s)$. Le taux d'acceptation est alors $1/M(s)$. La valeur de s qui maximise le taux d'acceptation est donc celle qui minimise la fonction M.

On commence par écrire une fonction qui calcule $M(s)$ en un point s donné.

```
> max.sachant.s<-function(s){
+   return(optimize(function(x){fgumbel(x)/dnorm(x,1,s)},
+     c(-100,100),maximum=TRUE)$objective)
+ }
```

Et on minimise sur s :

```
> (s<-optimize(max.sachant.s,c(0,100))$minimum)
```

```
[1] 8.625175
```

```
> (M<-max.sachant.s(s))
```

```
[1] 3.976797
```

Nous pouvons maintenant écrire l'algorithme d'acceptation-rejet demandé.

```
> fAR<-function(n,s,M){
+   res=rep(0,n)
+   count=0
+   for(i in 1:n){
+     x=rnorm(1,1,s)
+     count=count+1
+     while(fgumbel(x)<dnorm(x,1,s)*M*runif(1)){
+       x=rnorm(1,1,s)
+       count=count+1
+     }
+     res[i]=x
+   }
+   taux_acceptation<-n/count
+   return(list(taux=taux_acceptation,vecteur=res))
+ }
> fAR(10,s,M)

$taux
[1] 0.1960784

$vecteur
 [1] 10.7163597 -0.8090144  3.5658662  1.6134595  0.7890389  0.3332098  4.4368744  3.9389047
 [9]  3.4018856 -1.1918945
```

1.2 Question 2

On commence par calculer la fonction de répartition de la loi de Gumbel.
On peut vérifier que la fonction de répartition est

$$F(x) = \exp\left(-e^{-(x-\mu)/\beta}\right)$$

qui s'inverse aisément : pour $y \in]0,1[$,

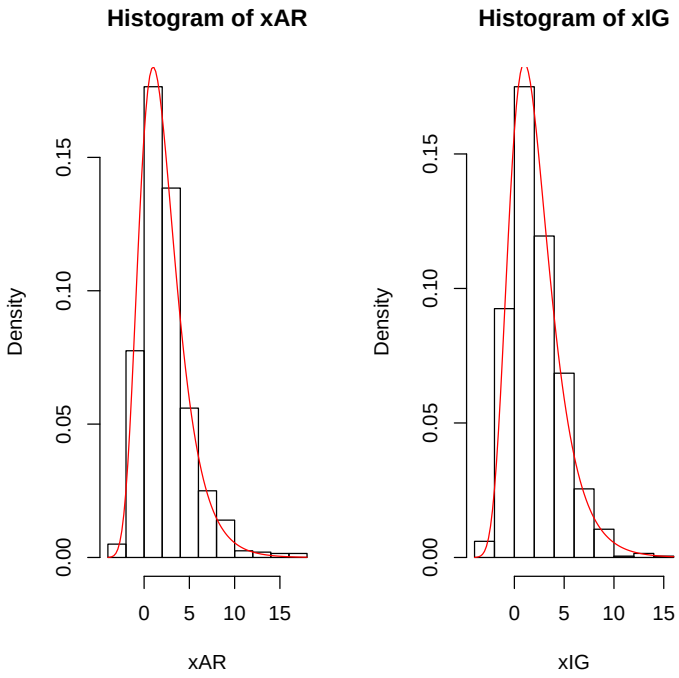
$$F^{-1}(y) = \mu - \beta \ln\left(-\ln(y)\right).$$

Le code correspondant :

```
> fIG<-function(n){
+   return(mu-beta*log(-log(runif(n))))
+ }
> fIG(10)

 [1]  2.0106617  5.2835848 -1.7312347  0.7498077  1.8109149  0.7020876  2.7208668  0.8532100
 [9]  2.0693050  1.7099939
```

1.3 Question 3

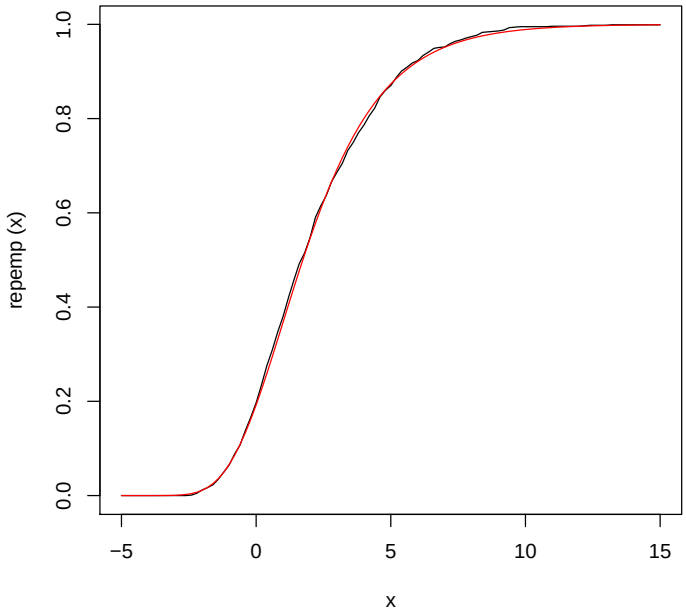
```
> n<-1000
> xAR<-fAR(n,s,M)$vecteur
> xIG<-fIG(n)
> par(mfrow=c(1,2))
> hist(xAR,prob=T)
> curve(fgumbel,add=T,col=2)
> hist(xIG,prob=T)
> curve(fgumbel,add=T,col=2)
```



1.4 Question 4

```
> par(mfrow=c(1,1))
> #Fonction de répartition empirique
> repemp<-function(x,vec=xIG){
+   n<-length(x)
+   res<-rep(0,n)
+   for(i in 1:n){
+   res[i]<-sum(vec<x[i])/length(vec)
+   }
+   return(res)
```

```
+ }
> #Fonction de répartition théorique
> reptheor<-function(x){
+   z<-(x-mu)/beta
+   return(exp(-exp(-z)))
+ }
> curve(repemp,-5,15)
> curve(reptheor,add=T,col=2)
```



1.5 Question 5

Pour choisir l'algorithme le plus avantageux, on évalue le temps nécessaire à la simulation d'un échantillon de taille n :

```
> system.time(fAR(n,s,M))

   user  system elapsed
  0.052   0.000   0.054

> system.time(fIG(n))

   user  system elapsed
      0       0       0
```

L'algorithme d'inversion générique est plus rapide.

Estimation de l'espérance et de la variance :

```
> mean(xIG)

[1] 2.113943

> var(xIG)

[1] 6.266904
```

Non demandé : les valeurs théoriques de l'espérance et de la variance sont

$$E[X] = \mu + \beta\gamma$$

$$Var(X) = \frac{\pi^2}{6}\beta^2$$

où γ est la constante d'Euler-Mascheroni. Les valeurs exactes sont donc

```
> mu+beta*(-digamma(1))

[1] 2.154431

> pi^2/6*beta^2

[1] 6.579736
```

ce qui est proche des estimations ci-dessus.

2 Exercice 2

Remarquons que l'intégrande est la densité d'une loi $N(0,1)$. On pourra donc utiliser la fonction `dnorm`.

2.1 Question a

La méthode de Monte-Carlo reposant sur une loi $N(0,1)$ est vouée à l'échec parce qu'une proportion trop faible des réalisations tombera dans l'intervalle $[4, +\infty[$:

```
> x<-rnorm(10^6)
> sum(x>4)/10^6

[1] 3.9e-05
```

Avec un taux d'acceptation de 3.9e-05, simuler suffisamment de données pour avoir un échantillon de taille raisonnable dans l'intervalle $[4, +\infty[$ prendrait trop longtemps.

2.2 Question b

On utilise de l'échantillonnage d'importance pour estimer I :

```
> n<-10^4
> x<-rexp(n)+4
> ichap<-mean(dnorm(x)/dexp(x-4))
> ichap

[1] 3.106769e-05
```

Pour produire un intervalle de confiance, on effectue 1000 estimations de I :

```
> confint<-function(n,alpha,lambda=1){
+   res<-rep(0,1000)
+   for(i in 1:1000){
+     x<-rexp(n,lambda)+4
+     res[i]<-mean(dnorm(x)/dexp(x-4,lambda))
+   }
+   return(quantile(res,c(alpha/2,1-alpha/2)))
+ }
> (IC1<-confint(n,.05) )

        2.5%        97.5%
3.087206e-05 3.240674e-05
```

(La fonction ci-dessus prend un paramètre `lambda` qui n'est pas utilisé dans cette question, mais sera utile dans la question **d**.)

2.3 Question c

Pour estimer λ, on a simplement besoin d'estimer le numérateur, puisqu'on a déjà une estimation du dénominateur. Le numérateur peut lui aussi être estimé par échantillonnage d'importance.

```
> (lambdachap<-mean(x*dnorm(x)/dexp(x-4))/ichap)

[1] 4.227867
```

2.4 Question d

Dans cette question, il est en fait plus efficace de prendre comme paramètre de l'exponentielle $1/(\hat{\lambda}-4)$ plutôt que $1/\hat{\lambda}$ donné dans l'énoncé. (On ne pénalise pas le fait d'avoir suivi l'énoncé.)

```
> (IC2<-confint(n,.05,1/(lambdachap-4)))

        2.5%        97.5%
3.164542e-05 3.169848e-05
```

Pour comparer les deux intervalles de confiance, on compare les longueurs :

```
> IC1[[2]]-IC1[[1]]

[1] 1.534679e-06

> IC2[[2]]-IC2[[1]]

[1] 5.30556e-08
```

On remarque que le second intervalle de confiance est bien plus resserré : la dernière méthode d'estimation est donc plus efficace.

Notons que si on suit l'énoncé, on obtient

```
> IC3 <- confint(n,.05,1/lambdachap)
> IC3[[2]] -IC3[[1]]

[1] 3.634422e-06
```

ce qui est nettement moins efficace. Si on a pris cette valeur pour le paramètre de l'exponentielle, on doit conclure que la première méthode est plus efficace.

Non demandé : on peut aussi obtenir la valeur de l'intégrale I directement avec la commande

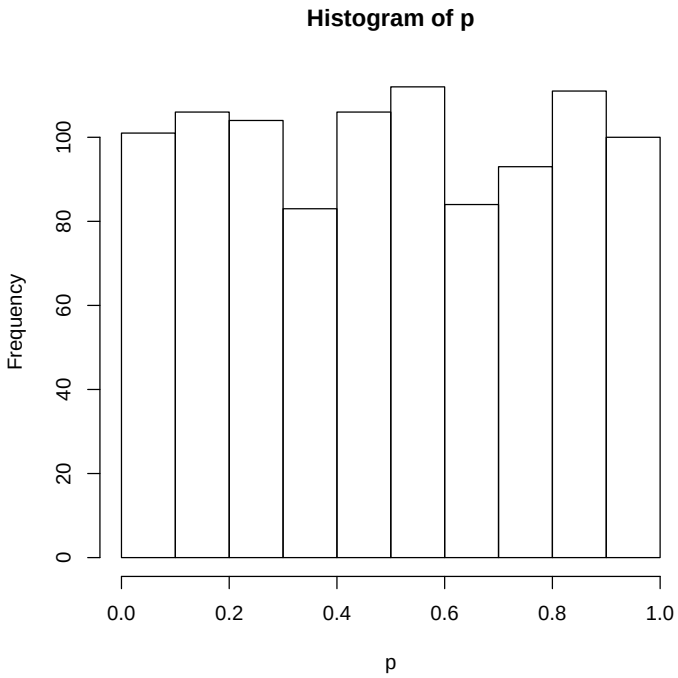
```
> 1-pnorm(4)

[1] 3.167124e-05
```

Cette valeur est bien dans tous les intervalles de confiance obtenus.

3 Exercice 3

3.1 Question a

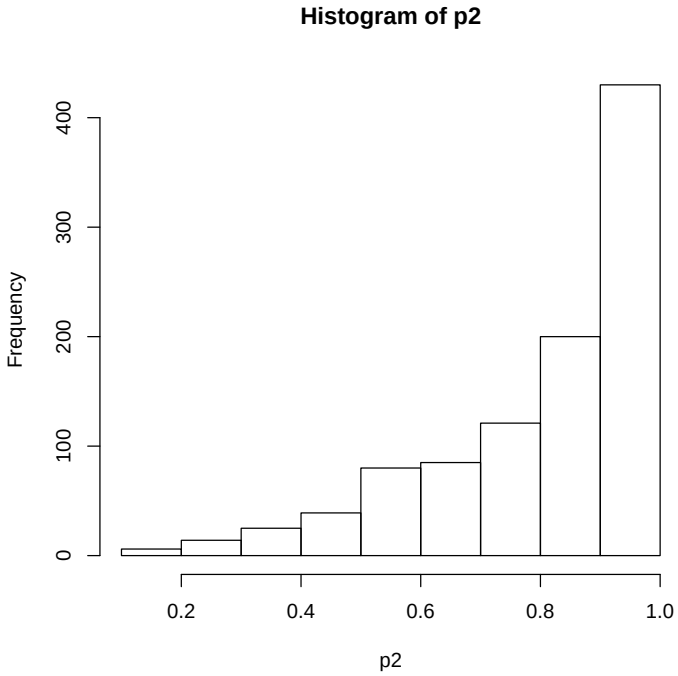
```
> nech<-1000
> p<-rep(0,nech)
> for(i in 1:nech){
+   x<-rnorm(50)
+   p[i]<-ks.test(x,"pnorm")$p.value
+ }
> hist(p)
> ks.test(p,"punif")

        One-sample Kolmogorov-Smirnov test

data:  p
D = 0.0219, p-value = 0.7224
alternative hypothesis: two-sided
```



La p-valeur étant supérieure à 5%, on accepte l'hypothèse d'uniformité du vecteur p.

3.2 Question b

```
> p2<-rep(0,nech)
> for(i in 1:nech){
+   x<-rnorm(50)
+   p2[i]<-ks.test(x,"pnorm",mean(x),sd(x))$p.value
+ }
> hist(p2)
> ks.test(p2,"punif")

        One-sample Kolmogorov-Smirnov test

data:  p2
D = 0.4558, p-value < 2.2e-16
alternative hypothesis: two-sided
```



La p-valeur étant très petite, on rejette l'hypothèse d'uniformité. Dans ce cas, le test n'est donc pas fiable : les p-valeurs devraient suivre une distribution uniforme. On a retrouvé le fait que lorsque les paramètres de la distribution sont estimés, la distribution de la statistique de test n'est pas la même.

3.3 Question c

Rappelons que la statistique du teste de Kolmogorov-Smirnov est

$$D(F, \hat{F}_n) = \max_{i=1,\dots,n} \left\{ \left| F(X_{(i)}) - \frac{i}{n} \right|, \left| F(X_{(i)}) - \frac{i-1}{n} \right| \right\}$$

où n est la taille de l'échantillon, F est la fonction de répartition de la distribution à laquelle on teste l'adéquation et $X_{(i)}$ est la $i$ème valeur de l'échantillon trié dans l'ordre croissant.

Notons F_1 la fonction de répartition de $N(\bar{x}, \hat{\sigma})$ et F_0 la fonction de répartition de $N(0,1)$.

Alors $\forall t \in \mathbb{R}, F_1(t) = F_0(\frac{t-\bar{x}}{\hat{\sigma}})$.

Or les deux tests qu'on nous demande de comparer consistent pour l'un à considérer $F_1(x_{(i)})$ et pour l'autre $F_0(\frac{x_{(i)}-\bar{x}}{\hat{\sigma}})$. Ces deux tests sont donc équivalents.

Tout test d'adéquation à une loi $N(\mu, \sigma^2)$ peut ainsi être ramené à un test d'adéquation à la loi $N(0,1)$. On s'intéresse donc uniquement à la loi $N(0,1)$ dans la suite.

3.4 Question d

```
> d<-rep(0,nech)
> for(i in 1:nech){
+   x<-rnorm(50)
+   d[i]<-ks.test(x,"pnorm",mean(x),sd(x))$statistic
+ }
> quantile(d,.95)

      95%
0.1225547
```

Pour tester correctement la normalité d'un échantillon de taille 50 de loi normale de paramètres inconnus (donc estimés), on doit accepter l'hypothèse de normalité ssi

$$D < 0.1226$$

4 Exercice 4

```
> f<-function(x){
+   x*exp(-4*x^2/15)
+ }
```

4.1 Question 1

Pour faire de l'acceptation-rejet, on va chercher une densité g telle que pour une certaine constante M, $\forall x \in \mathbb{R}_+, f(x) < CMg(x)$.

Prenons pour g la densité de la loi $N(0, 15/4)$ restreinte à $\mathbb{R}_+$. On montre que la condition $f < CMg$ est vérifiée en s'assurant que $x \exp(-2x^2/15)$ est borné sur $\mathbb{R}_+$, ce qui se fait facilement.

```
> sigma<-sqrt(15/4)
> g<-function(x){
+   2*dnorm(x,0,sigma) # Le facteur 2 provient de la restriction à R+
+ }
> (M<-optimize(function(x){f(x)/g(x)},c(0,100),maximum=T)$objective)

[1] 2.85065
```

Notons que simuler selon g revient à simuler une réalisation de la loi $N(0, 15/4)$ et à prendre la valeur absolue.

On peut maintenant écrire l'algorithme d'acceptation-rejet :

```
> fAR1 <-function(n){
+   count<-0 # compteur pour déterminer le taux d'acceptation
+   res=rep(0,n)
+   for(i in 1:n){
```

```
+     x<-abs(rnorm(1,0,sigma))
+     count<-count+1
+     while(f(x)<g(x)*M*runif(1)){
+       x<-abs(rnorm(1,0,sigma))
+       count<-count+1
+     }
+   res[i]<-x
+   }
+
+   return(list(vec=res,taux=n/count))
+
+ }
```

4.2 Question 2

Le taux d'acceptation est $1/CM$. On obtient donc une estimation de C comme suit :

```
> n<-10^4
> temp<-fAR1(n)
> ech<-temp$vec
> taux<-temp$taux
> (Cchap<-1/(M*taux))
```

```
[1] 0.5302299
```

Comparons à la valeur trouvée par intégration numérique :

```
> 1/integrate(f,0,Inf)$value
```

```
[1] 0.5333333
```

Calcul de l'intervalle de confiance : le taux d'acceptation observé est $\frac{n}{k}$ où k est le nombre de simulations selon g (variable `count`). Le taux d'acceptation théorique est $1/CM$.

Un intervalle de confiance au seuil α pour $1/CM$ est donc $\frac{n}{k} \pm \frac{q_{1-\alpha/2}}{\sqrt{k}} \sqrt{\frac{n}{k}(1-\frac{n}{k})}$ (intervalle de confiance pour des données suivant une distribution de Bernoulli). Un intervalle de confiance pour C est donc

$$\left[\frac{1}{M\left(\frac{n}{k} + \frac{q_{1-\alpha/2}}{\sqrt{k}} \sqrt{\frac{n}{k}(1-\frac{n}{k})}\right)}, \frac{1}{M\left(\frac{n}{k} - \frac{q_{1-\alpha/2}}{\sqrt{k}} \sqrt{\frac{n}{k}(1-\frac{n}{k})}\right)} \right].$$

Le nombre k n'est pas accessible directement, mais on a accès à n et $taux$: dans la formule ci-dessous, on utilise donc $n/k = taux$ et $k = n/taux$.

```
> c(1/(M*(taux+pnorm(.975)/sqrt(n/taux)*sqrt(taux*(1-taux)))),
+   1/(M*(taux-pnorm(.975)/sqrt(n/taux)*sqrt(taux*(1-taux)))))
```

```
[1] 0.5276661 0.5328187
```

Autre méthode : On pouvait aussi calculer un intervalle de confiance en effectuant 1000 estimations de C, comme à la question 6.

4.3 Question 3

La fonction de répartition F de X est une primitive de f. Elle est donc de la forme

$$F(x) = K - \frac{15}{8} C \exp\left(\frac{-4x^2}{15}\right)$$

avec $K \in \mathbb{R}$.

La condition $\lim_{x\to\infty} F(x) = 1$ implique que $K = 1$. La condition $F(0) = 0$ implique que $C = 8/15$; on peut d'ailleurs vérifier que cela correspond bien aux valeurs trouvées par estimation et par intégration numérique à la question précédente :

```
> 8/15
```

```
[1] 0.5333333
```

Cette fonction s'inverse en

$$F^{-1}(y) = \sqrt{-\frac{15}{4}\ln(1-y)}.$$

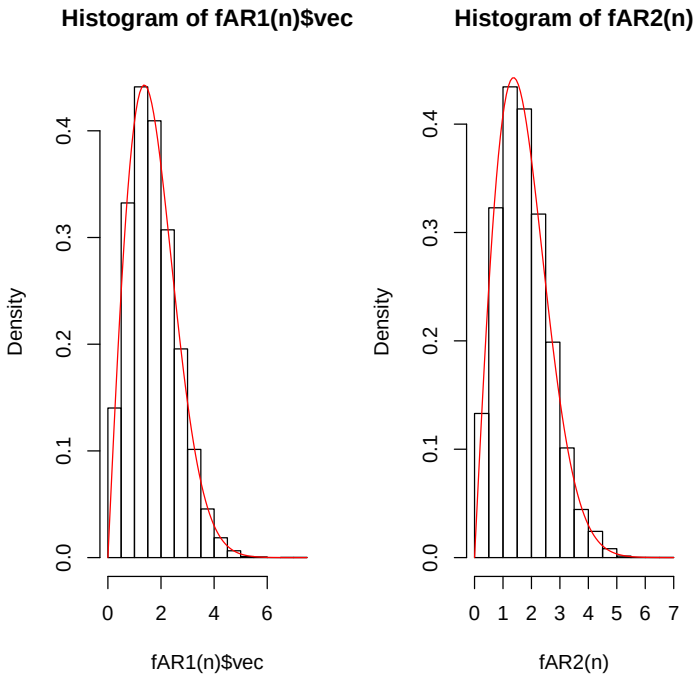
D'où l'algorithme de simulation par inversion générique :

```
> fAR2<-function(n){
+   y<-runif(n)
+   return(sqrt(-15/4*log(1-y)))
+ }
```

Autre méthode : On pouvait aussi utiliser l'estimation de C obtenue à la question précédente. Dans ce cas, l'algorithme de génération par inversion générique n'est pas exact. Il faudrait alors préférer l'acceptation-rejet à la question 5.

4.4 Question 4

```
> par(mfrow=c(1,2))
> hist(fAR1(n)$vec,prob=T)
> curve(f(x)*8/15,add=T,col=2)
> hist(fAR2(n),prob=T)
> curve(f(x)*8/15,add=T,col=2)
```



4.5 Question 5

La méthode par inversion générique est plus rapide ; c'est donc celle-là qu'on préférera.

```
> system.time(fAR1(n))

   user  system elapsed
  0.208   0.000   0.209

> system.time(fAR2(n))

   user  system elapsed
  0.004   0.000   0.001
```

4.6 Question 6

Estimation ponctuelle :

```
> ech<-fAR2(n)
> mean(ech) # estimation de E[X]

[1] 1.725916
```

```
> mean(ech^2) # estimation de E[X^2]

[1] 3.770806

> mean(ech>2) # estimation de P[X>2]

[1] 0.3507
```

Intervalles de confiance :

```
> a<-b<-c<-rep(0,1000)
> for(i in 1:1000){
+   ech<-fAR2(n)
+   a[i]<-mean(ech) # estimation de E[X]
+   b[i]<-mean(ech^2) # estimation de E[X^2]
+   c[i]<-mean(ech>2) # estimation de P[X>2]
+ }
> quantile(a,c(.025,.975))

    2.5%    97.5%
1.699693 1.734134

> quantile(b,c(.025,.975))

    2.5%    97.5%
3.678932 3.823987

> quantile(c,c(.025,.975))

     2.5%     97.5%
0.3351975 0.3536025
```

Non demandé : la loi de X s'appelle la loi de Rayleigh, de paramètre $\sigma = \sqrt{15/8}$. On peut comparer les valeurs trouvées ci-dessus aux valeurs théoriques pour cette loi.

```
> sigma<-sqrt(15/8)
> sigma*sqrt(pi/2) #E[X]

[1] 1.716171

> 2*sigma^2 #E[X^2]

[1] 3.75

> exp(-4*2^2/15) #P[X>2]

[1] 0.3441538
```

5 Exercice 5

5.1 Question a

```
> n<-100
> alpha<-.11
> x<-c(rnorm(100,sd=10),rcauchy(50),rexp(20,5))
> ks.test(x,"pnorm",mean(x),sd(x))

        One-sample Kolmogorov-Smirnov test

data:  x
D = 0.3802, p-value < 2.2e-16
alternative hypothesis: two-sided
```

On rejette l'hypothèse de normalité des données.

.

5.2 Question b

```
> qchap=quantile(x,alpha)
> qchapboot=rep(0,2500)
> for(i in 1:2500){
+   xstar<-sample(x,length(x),rep=T)
+   qchapboot[i]<-quantile(xstar,alpha)
+ }
> deltastar<-sqrt(n)*(qchapboot-qchap)
```

5.3 Question c

La moyenne de la distribution normale est fixée à 0, mais la variance est libre.

La commande suivante génère une erreur à cause d'ex aequo :

```
> ks.test(deltastar,"pnorm",0,sd(deltastar))

        One-sample Kolmogorov-Smirnov test

data:  deltastar
D = 0.108, p-value < 2.2e-16
alternative hypothesis: two-sided

> warnings()

Messages d'avis :
1: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
2: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
```

```
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
3: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
4: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
5: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
6: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
7: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
8: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
9: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
10: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
11: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
12: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
13: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
14: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
15: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
16: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
17: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
18: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
19: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
20: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
21: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
22: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
23: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
24: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
25: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
```

```
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
26: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
27: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
28: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
29: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
30: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
31: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
32: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
33: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
34: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
35: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
36: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
37: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
38: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
39: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
40: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
41: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
42: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
43: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
44: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
45: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
46: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
47: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
48: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
```

```
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
49: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
50: In D[i] = (Y1 - mean(Y1))/sd(Y1) :
  le nombre d'objets à remplacer n'est pas multiple de la taille du remplacement
```

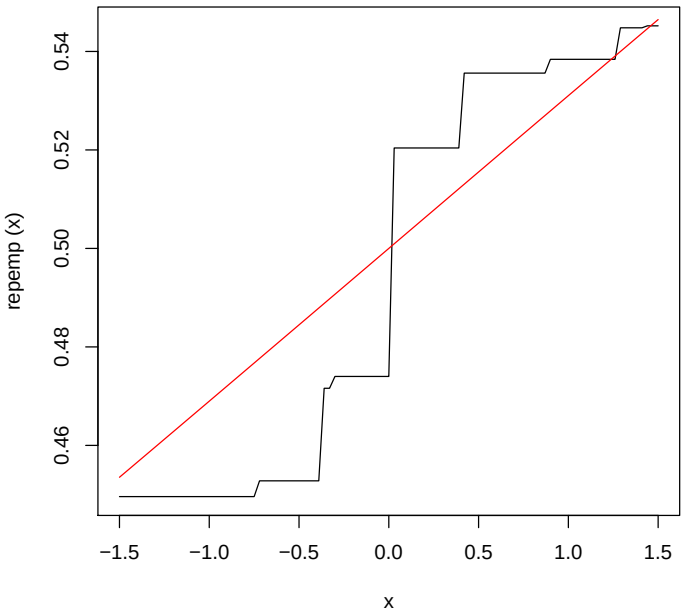
On utilise donc

```
> ks.test(jitter(deltastar),"pnorm",0,sd(deltastar))

        One-sample Kolmogorov-Smirnov test

data:  jitter(deltastar)
D = 0.1074, p-value < 2.2e-16
alternative hypothesis: two-sided
```

On accepte donc l'hypothèse d'un loi normale centrée (au niveau 95%).

5.4 Question d

```
> repemp<-function(x,vec=deltastar){
+   n<-length(x)
+   res<-rep(0,n)
+   for(i in 1:n){
+   res[i]<-sum(vec<x[i])/length(vec)
+   }
+   return(res)
+ }
> curve(repemp,-1.5,1.5)
> curve(pnorm(x,0,sd(deltastar)),add=T,col=2)
```



6 Exercice 6

```
> p<-.6
> f<-function(x){
+   return(p*dnorm(x,0,1)+(1-p)*dnorm(x,3,1))
+ }
> nech<-1000
> Z<-replicate(nech,rnorm(1,sample(c(0,3),prob=c(p,1-p)),1))
```

6.1 Question a

Estimation de $P = P[X < 2]$.

6.1.1 Question a1

Par intégration numérique :

```
> integrate(f,-Inf,2)

0.649812 with absolute error < 9.5e-07
```

6.1.2 Question a2

Soit Y une variable aléatoire de distribution $Bin(p)$. Pour tirer une réalisation de la variable aléatoire de densité f, on peut poser que si $Y = 1$ alors $X \sim N(0,1)$ et si $Y = 0$ alors $X \sim N(3,1)$.

Alors $P[X < 2] = P[X < 2|Y = 0]P[Y = 0] + P[X < 2|Y = 1]P[Y = 1]$.

On obtient ainsi la valeur exacte de P :

```
> p*pnorm(2,0,1)+(1-p)*pnorm(2,3,1)

[1] 0.649812
```

6.2 Question b

Estimation de P :

```
> (pchap<-mean(Z<2))

[1] 0.664
```

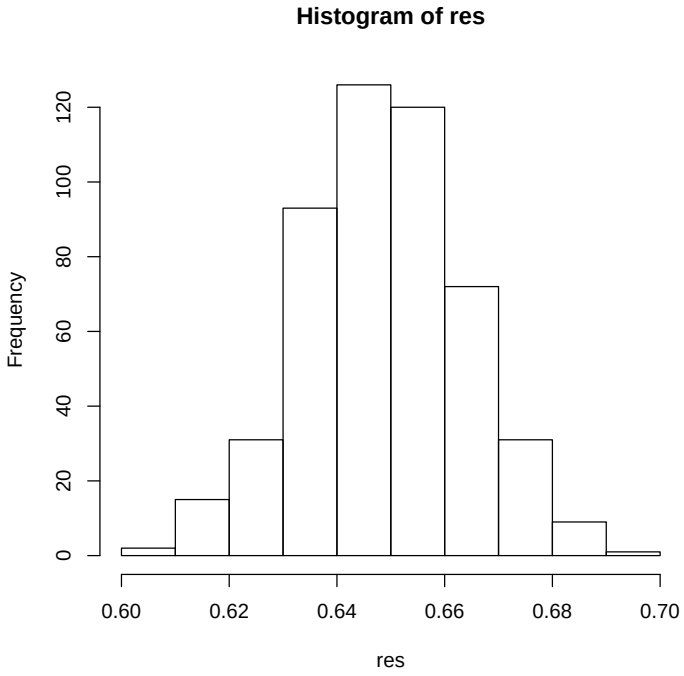
Pour l'intervalle de confiance, on utilise la formule vue en cours :

$$\hat{P} \pm q_{1-\alpha/2} \frac{1}{\sqrt{n}} \sqrt{\hat{P}(1-\hat{P})}$$

```
> r<-pnorm(.975)/sqrt(nech)*sqrt(pchap*(1-pchap))
> (confint<-pchap+c(-r,r))

[1] 0.6515246 0.6764754
```

6.3 Question c

```
> res<-rep(0,500)
> for(i in 1:500){
+   Z<-replicate(nech,rnorm(1,sample(c(0,3),prob=c(p,1-p)),1))
+   res[i]<-mean(Z<2)
+ }
> hist(res)
> quantile(res,c(.025,.975))

    2.5%    97.5%
0.619475 0.678000
```



6.4 Question d

```
> ks.test(jitter(res),"pnorm",mean(res),sd(res))

        One-sample Kolmogorov-Smirnov test

data:  jitter(res)
D = 0.0234, p-value = 0.9478
alternative hypothesis: two-sided
```

On accepte donc l'hypothèse de normalité.