TP 3

Gautier Poursin

13/03/2020

Maximum de vraisemblance

1. Ajuster une loi Bernoulli

a.

p correspond à la proba P(X=1). Il faut donc faire plusieurs échantillons puis ensuite faire la moyenne des résultats obtenus pour obtenir p.

b.

```
bern<-rbinom (n=10,p=0.7, size=1)

L_bern <- function (bern, p=0.7){
  produit = (p**sum(bern)*(1-p)**(length(bern)-sum(bern)))
}

Vraisemblance=L_bern(bern)
print(bern)
```

```
##  [1] 1 1 1 0 1 0 1 0 0 1
```

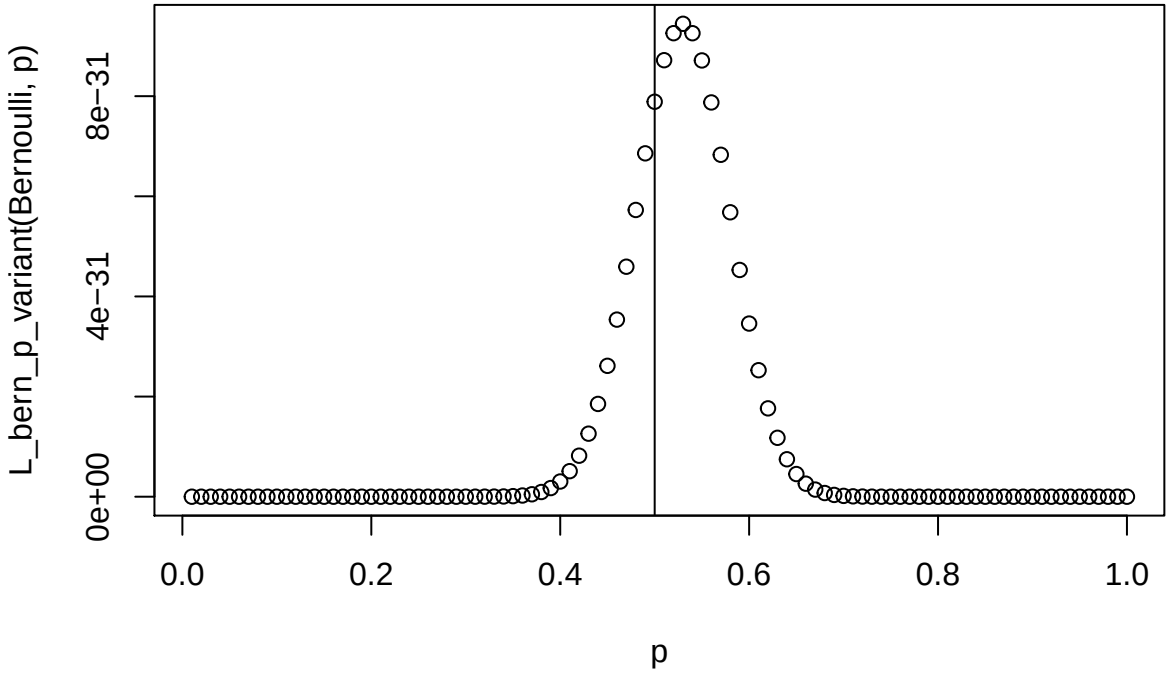
```
print(Vraisemblance)
```

```
## [1] 0.0009529569
```

c.

```
##   [1] 6.235254e-107  3.485101e-91  4.631248e-82  1.191043e-75  9.963678e-71
##   [6]  9.528519e-67  2.036548e-63  1.451434e-60  4.464763e-58  7.069650e-56
##  [11]  6.533451e-54  3.866265e-52  1.571819e-50  4.636744e-49  1.036397e-47
##  [16]  1.817546e-46  2.573139e-45  3.010883e-44  2.969865e-43  2.510841e-42
##  [21]  1.845423e-41  1.193571e-40  6.864982e-40  3.543518e-39  1.654626e-38
##  [26]  7.038470e-38  2.744441e-37  9.862863e-37  3.282738e-36  1.016329e-35
##  [31]  2.938171e-35  7.959169e-35  2.026537e-34  4.863485e-34  1.102895e-33
##  [36]  2.368598e-33  4.827221e-33  9.352765e-33  1.725546e-32  3.035906e-32
##  [41]  5.100221e-32  8.190833e-32  1.258770e-31  1.852802e-31  2.614024e-31
```

```
##  [46]  3.537299e-31  4.593591e-31  5.727205e-31  6.857890e-31  7.888609e-31
##  [51]  8.718366e-31  9.257961e-31  9.445401e-31  9.257364e-31  8.713862e-31
##  [56]  7.874831e-31  6.829435e-31  5.680680e-31  4.528954e-31  3.458087e-31
##  [61]  2.526491e-31  1.764366e-31  1.176333e-31  7.477495e-32  4.524869e-32
##  [66]  2.602182e-32  1.419449e-32  7.328606e-33  3.572740e-33  1.640194e-33
##  [71]  7.069658e-34  2.851345e-34  1.072034e-34  3.741351e-35  1.206222e-35
##  [76]  3.573129e-36  9.665331e-37  2.370718e-37  5.230494e-38  1.028440e-38
##  [81]  1.782893e-39  2.691175e-40  3.485284e-41  3.805760e-42  3.431565e-43
##  [86]  2.491355e-44  1.412076e-45  6.013124e-47  1.832848e-48  3.757102e-50
##  [91]  4.770806e-52  3.357254e-54  1.119964e-56  1.408916e-59  4.687500e-63
##  [96]  2.276119e-67  5.291769e-73  4.823828e-81  5.870368e-95  0.000000e+00
```



Ici, pour estimer le paramètre, on doit maximiser la vraisemblance obtenue pour déterminer le paramètre. On a tracé pour p=0,5 et le maximu de vraisemblance est atteint en $p^* \simeq 0.5$

d.

```r
optimze_bernoulli<-function(p){
  Bernoulli=rbinom(n=100,size=1, p=0.5)
  resul=L_bern_p_variant(Bernoulli,p)
}

max_optimize=optimize(optimze_bernoulli, rep(1:100)/100, maximum=TRUE)
print(max_optimize)
```

```
## $maximum
## [1] 0.5731198
##
## $objective
## [1] 2.676523e-31
```

Le maximum obtenu avecla fonction maximize est bien similaire à notre paramètre p.

e.

```
optimze_bernoulli_20<-function(p){
  B20=rbinom(n=20,p=0.5,size=1)
  resul=L_bern_p_variant(B20,p)
}

max_optimize_20=optimize(optimze_bernoulli_20, rep(1:100)/100, maximum=TRUE)
print(max_optimize_20)
```

```
## $maximum
## [1] 0.3751223
##
## $objective
## [1] 6.504148e-08
```

```
optimze_bernoulli_100<-function(p){
  B100=rbinom(n=100,p=0.5,size=1)
  resul=L_bern_p_variant(B100,p)
}

max_optimize_100=optimize(optimze_bernoulli_100, rep(1:100)/100, maximum=TRUE)
print(max_optimize_100)
```

```
## $maximum
## [1] 0.5113065
##
## $objective
## [1] 6.416741e-31
```

```
optimze_bernoulli_500<-function(p){
  B500=rbinom(n=500,p=0.5,size=1)
  resul=L_bern_p_variant(B500,p)
}

max_optimize_500=optimize(optimze_bernoulli_500, rep(1:100)/100, maximum=TRUE)
print(max_optimize_500)
```

```
## $maximum
## [1] 0.5099061
##
## $objective
## [1] 3.956162e-151
```

```r
optimze_bernoulli_1000<-function(p){
  B1000=rbinom(n=1000,p=0.5,size=1)
  resul=L_bern_p_variant(B1000,p)
}

max_optimize_1000=optimize(optimze_bernoulli_1000, rep(1:100)/100, maximum=TRUE)
print(max_optimize_1000)
```

```
## $maximum
## [1] 0.504648
##
## $objective
## [1] 8.297362e-302
```

```r
optimze_bernoulli_1500<-function(p){
  B1500=rbinom(n=1500,p=0.5,size=1)
  resul=L_bern_p_variant(B1500,p)
}

max_optimize_1500=optimize(optimze_bernoulli_1500, rep(1:100)/100, maximum=TRUE)
print(max_optimize_1500)
```

```
## $maximum
## [1] 0.9999348
##
## $objective
## [1] 0
```

```r
optimze_bernoulli_2000<-function(p){
  B2000=rbinom(n=2000,p=0.5,size=1)
  resul=L_bern_p_variant(B2000,p)
}

max_optimize_2000=optimize(optimze_bernoulli_2000, rep(1:100)/100, maximum=TRUE)
print(max_optimize_2000)
```

```
## $maximum
## [1] 0.9999348
##
## $objective
## [1] 0
```

```r
ecart<- as.numeric(c(max_optimize_20,max_optimize_100,max_optimize_500,max_optimize_1000,max_optimize_15

for (i in 1:length(ecart)) ecart[i]=ecart[i]-0.5
paste("l'écart pour n= 20 est ", ecart[1],sep="")
```

```
## [1] "l'écart pour n= 20 est -0.124877710183857"
```

```r
paste("l'écart pour n= 100 est ", ecart[3],sep="")
```

```
## [1] "l'écart pour n= 100 est 0.0113065359424229"
```

```r
paste("l'écart pour n= 500 est " ,ecart[5],sep="")
```

```
## [1] "l'écart pour n= 500 est 0.00990606140173789"
```

```r
paste("l'écart pour n= 1000 est ", ecart[7],sep="")
```

```
## [1] "l'écart pour n= 1000 est 0.00464800237529239"
```

```r
paste("l'écart pour n= 1500 est ",ecart[9],sep="")
```

```
## [1] "l'écart pour n= 1500 est 0.499934811326493"
```

```r
paste("l'écart pour n= 2000 est ",ecart[11],sep="")
```

```
## [1] "l'écart pour n= 2000 est 0.499934811326493"
```

on peut donc conclure qu'à partir d'un certain n, la précision n'est plus bonne. Cela est du à l'instabilité numérique, qui augmente au fur et a mesure qu'il y ait des calculs. Pour éviter cette instabilité, il est possible d'utiliser la fonction logarithme.

f.

```r
Distribinc<-write.csv("distribution_inconue_2_100_realisations.csv")
```

```
## "","x"
## "1","distribution_inconue_2_100_realisations.csv"
```

```r
distrib<-read.csv("distribution_inconue_2_100_realisations.csv", header=TRUE)

optimze_bernoulli<-function(p){
  resul=L_bern_p_variant(distrib[2],p)
  return(resul)
}

mean_distrib<-sum(distrib[2])/100
paste("la moyenne de cette distribution inconnue est ", mean_distrib,sep="")
```

```
## [1] "la moyenne de cette distribution inconnue est 0.58"
```

```r
max_optimize_distrib=optimize(optimze_bernoulli, rep(1:100)/100, maximum=TRUE)
print(max_optimize_distrib)
```

```
## $maximum
## [1] 0.9999348
##
## $objective
## [1] 3.895343e+238
```

Le maximum obtenu ici vaut 0.99, mais ce maximum n'est pas valide. En effet, la moyenne vaut 0.58 et l'estimateur devrait se rapprocher de cette valeur.

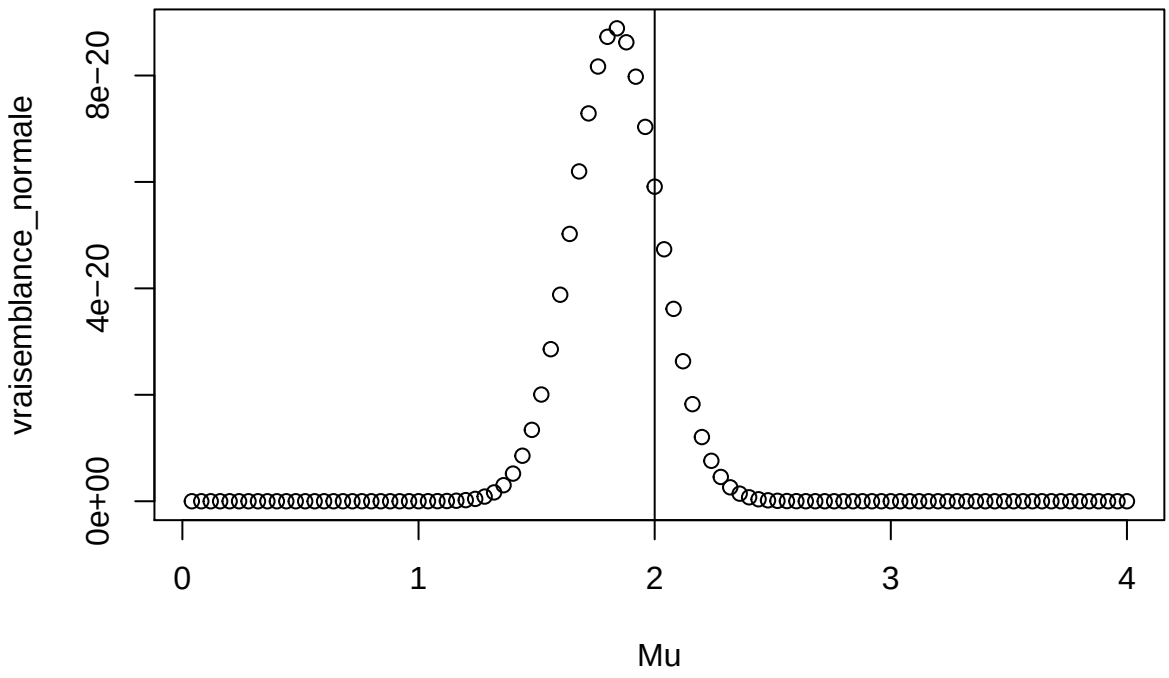
ajuster une loi normale d'écart connu

a.

```
mu=2
sigma=1
N=30

Normale<-rnorm(N,mu,sigma)

L_norm<-function (mu,sigma,X){
  produit=1
  for (i in 1:length(X)) produit=produit*dnorm(X[i],mean=mu,sd=sigma)
  return(produit)
}
```

b.

```
Mu=rep(1:100)

for (i in 1:100) Mu[i]=i/100*4
vraisemblance_normale=L_norm(Mu,sigma,Normale)

plot(Mu,vraisemblance_normale)
abline(v=2)
```



On retrouve bien que le maximum de vraisemblance est atteint pour mu=2.

c.

```
max_optimize_normale=optimize(function(mu) {L_norm(mu,sigma,Normale)}, 4*rep(1:100)/100, maximum=TRUE)
print(max_optimize_normale[1])
```

```
## $maximum
## [1] 1.835075
```

d.

```
for (N in c(10,100,500,1000,1500,2000)){
  normale=rnorm(N,mu,sigma)
  max_optimize_normale=optimize(function(mu) {L_norm(mu,sigma,normale)}, 4*rep(1:100)/100, maximum=TRUE
  print(N)
  print(max_optimize_normale[1])
}
```

```
## [1] 10
## $maximum
```

```
## [1] 2.209643
##
## [1] 100
## $maximum
## [1] 1.931574
##
## [1] 500
## $maximum
## [1] 3.999941
##
## [1] 1000
## $maximum
## [1] 3.999941
##
## [1] 1500
## $maximum
## [1] 3.999941
##
## [1] 2000
## $maximum
## [1] 3.999941
```

```
Log_norm<-function (mu,sigma,X){
  somme=0
  for (i in 1:length(X)) somme=somme+ log(dnorm(X[i],mean=mu,sd=sigma))
  return(somme)
}

for (N in c(10,100,500,1000,1500,2000)){
  normale=rnorm(N,mu,sigma)
  max_optimize_normale=optimize(function(mu) {Log_norm(mu,sigma,normale)}, 4*rep(1:100)/100, maximum=TRU
  print(N)
  print(max_optimize_normale[1])
}
```

```
## [1] 10
## $maximum
## [1] 1.809663
##
## [1] 100
## $maximum
## [1] 2.021039
##
## [1] 500
## $maximum
## [1] 1.991112
##
## [1] 1000
## $maximum
## [1] 1.975628
##
## [1] 1500
## $maximum
## [1] 1.984656
```

```
##
## [1] 2000
## $maximum
## [1] 2.026159
```

On voit bien qu'en utilisantle logarithme, on a supprimé ces erreurs d'instabilités.

Ajuster une loi à plusieurs paramètres

1.

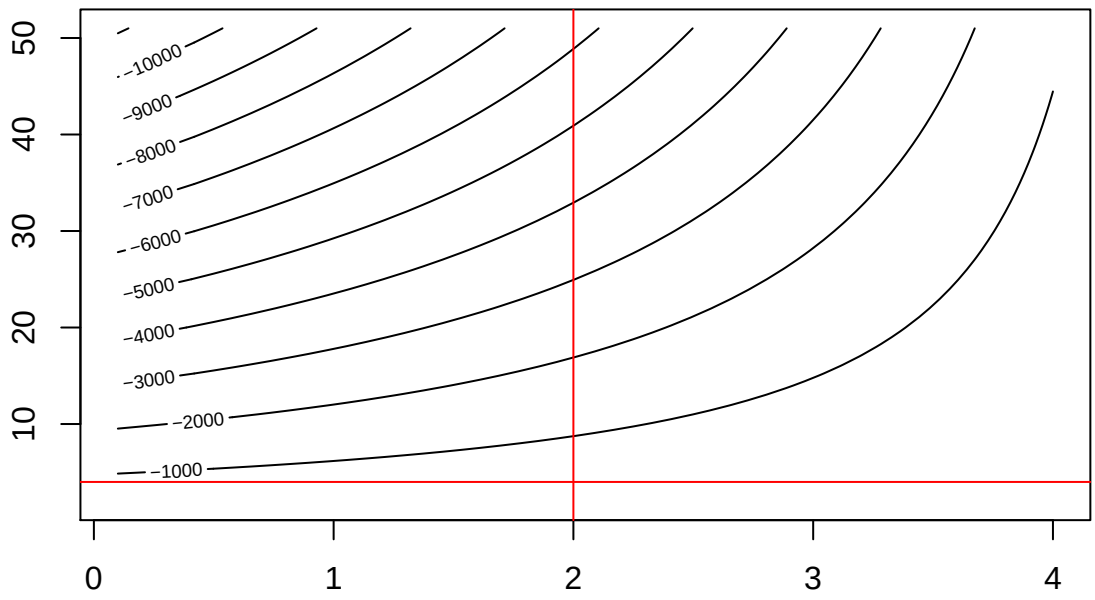
Voici a quoi ressemble la vraisemblance pour la loi exponentielle translatée :

$$\mathcal{L}(x_1, ..., x_n; (\lambda, L)) = \log(\prod_{i=1}^{n} \lambda e^{-\lambda(x_i - L)}) = n\log(\lambda) + n\lambda L - \lambda \sum_{i=1}^{n} x_i$$

```
lambda=2
L=4
echantillon_exp=rexp(n=50,rate=1)/lambda+L
Log_exp<-function (L,lambda){
  return(sum(log(lambda*exp(-lambda*(echantillon_exp-L)))))
}

library("stats4")
library("ggplot2")
```

```
## Warning: package 'ggplot2' was built under R version 3.6.3
```

```
lambda=seq(2,length.out = 50)
L=seq(0.1,4,length.out=50)
vraisemblance=matrix(1,50,50)

for (i in 1:50){
  for(j in 1:50)
    vraisemblance[i,j]=Log_exp(L[i],lambda[j])
}
contour(L,lambda,vraisemblance)
abline(v=2,h=4,col="red")
```

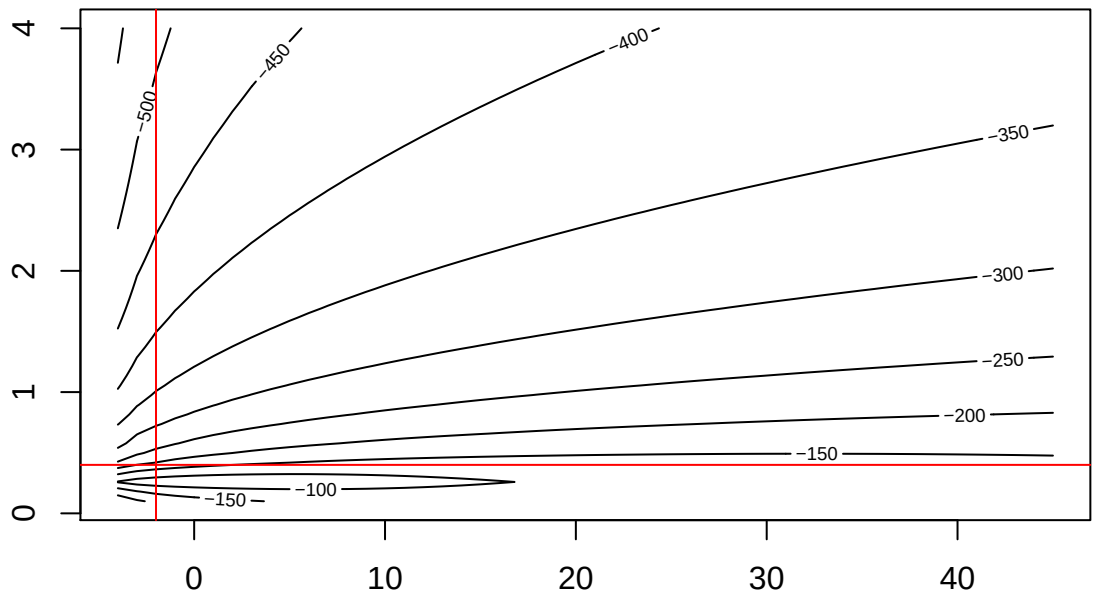


2.

Voici à quoi ressemble la vraisemblance pour la loi de Cauchy :

$$\mathcal{L}(x_1,...,x_n;(x_0,\alpha)) = \log(\prod_{i=1}^{n} \frac{1}{\pi}\frac{\alpha}{(x_i-x_0)^2+\alpha^2}) = n\log(\frac{\alpha}{\pi}) - \sum_{i=1}^{n}\log((x_i-x_0)^2+\alpha^2)$$

```
alpha=0.4
x0=-2
echantillon_cauchy=rcauchy(n=50,location=x0,scale=alpha)
Log_cauchy<-function(alpha,x0){
  return(sum(log((1/pi)*alpha/((echantillon_cauchy-x0)**2+alpha**2))))
}
x0=seq(-4.0,length.out = 50)
Alpha=seq(0.1,4,length.out=50)
vraisemblance=matrix(1,50,50)

for (i in 1:50){
  for(j in 1:50)
    vraisemblance[i,j]=Log_cauchy(Alpha[i],x0[j])
}

contour(x0,Alpha,vraisemblance)
abline(v=-2,h=0.4,col="red")
```



Utilisons le solveur mle de la librairie stats4.

```
minusLogCau <- function(alpha, x0){ return(-Log_cauchy(alpha, x0)); }

minusLogExp <- function(lambda, L){ return(-Log_exp(L,lambda)); }
mleCau = c(0, 0);
mleExp = c(0, 0);

mleCau = mle(minuslogl = minusLogCau, start = list(alpha =0.4, x0 =-2))
```

```
## Warning in log((1/pi) * alpha/((echantillon_cauchy - x0)^2 + alpha^2)):
## production de NaN
```

```
mleExp = mle(minuslogl = minusLogExp, start = list(lambda =4), fixed=(list(L= 4)))
```

```
## Warning in log(lambda * exp(-lambda * (echantillon_exp - L))): production de NaN

## Warning in log(lambda * exp(-lambda * (echantillon_exp - L))): production de NaN
```

```
print(mleCau)
```

```
##
## Call:
## mle(minuslogl = minusLogCau, start = list(alpha = 0.4, x0 = -2))
```

```
##
## Coefficients:
##     alpha        x0
##  0.389651 -1.949682
```

```
print(mleExp)
```

```
##
## Call:
## mle(minuslogl = minusLogExp, start = list(lambda = 4), fixed = (list(L = 4)))
##
## Coefficients:
##   lambda        L
## 1.868563 4.000000
```

Les résultats trouvés pour alpha et x0 sont satisfaisant, tandis que celui pour lambda ne l'est pas. On remarque un écart entre les 2 résultats.

Conclusion

Dans ce TP, nous avons vu la méthode des estimateurs de vraisemblance. Cela nous a permis de déterminer le maximum de vraisemblance pour l'échantillon fourni. Il y a cependant une erreur de code puisque celui ci renvoie 0.99(question f) alors que le maximum est de 0.58. Concernant la seconde partie, nous avons travaillé avec une loi normale d'écart type connu. Nous avons tracé la courbe et le maximum de vraisemblance correspond bien à la valeur de ntore paramètre. Puis, nous avons essayé d'éviter les instabilité causés par le calcul du maximum lorsque N devient grand. Pour cela, nous avons utlisé le logarithme, qui supprime bien les erreurs (voir question 2.d). Enfin, nous avons travaillé avec des lois à plusieurs paramètres. Les résultats obtenus par les contours ne m'ont pas permis de conclure quelque chose. Mais avec le mle, nous avons retrouvé les valeurs de alpha et xo.