CR_TP1

Matisse Landais

26 septembre 2018

Histogrammes et polygone de fréquences

Il faut déja créer un jeu de données avec des valeurs (discrètes ou continues), les trier puis créer les différents histogrammes.

```
x<-c(91.6,35.7,251.3,24.3,5.4,67.3,170.9,9.5,118.4,57.1)
y<-sort(x)
```

Nombre de classe

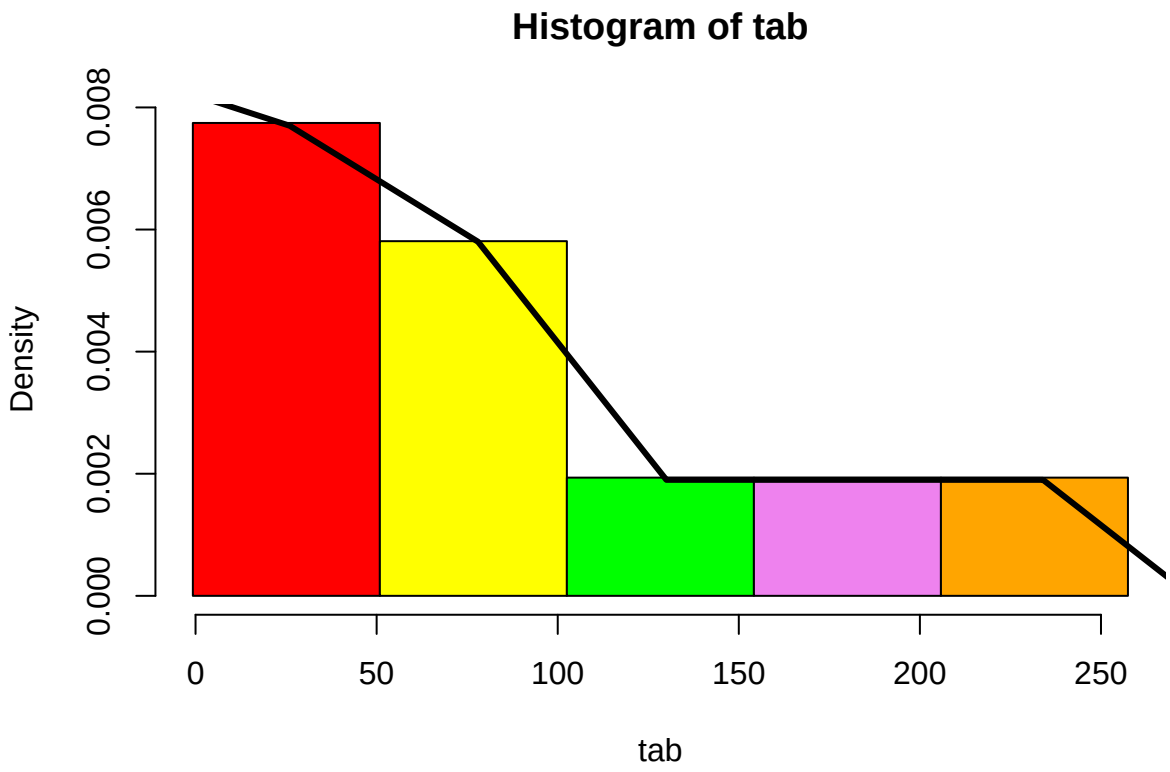
La règle de Sturges indique que le nombre de classe est entre 5 et 20. k le nombre de classes = 1 + lb(n) sachant que n est le nombre d'éléments dans le vecteur. Ici, n <= 22 donc k = 5.

Choix des bornes

Il faut maintenant choisir les bornes de a0 à ak respectant une homogénéité des largeurs des classes.

```
histogrammeClasse <-function(tab) {
  # Soit c le tableau que l'on veut plot.

  k = 0 # Nombre de classes
  a0 = 0 # Borne min
  ak = 0 # Borne max
  h = 0 # Hauteur du rectangle

  # Calcul du nombre de classes
  if (length(tab) <= 22) {
    k = 5
  }else{
    k = 1 + round(log2(length(tab)))
  }

  # Choix des bornes
  # Premier et dernier élément du table
  preElem = tab[1]
  derElem = tail(tab,n=1)

  # Premiere borne et dernière borne. 0 et 260 dans notre cas (ampoule)
  a0 = preElem - 0.025*(derElem - preElem)
  ak = derElem + 0.025*(derElem - preElem)

  # Calcul de la largeur
  h = (ak - a0)/k

  # On plot
  colors = c("red", "yellow", "green", "violet", "orange", "blue", "pink", "cyan")
```

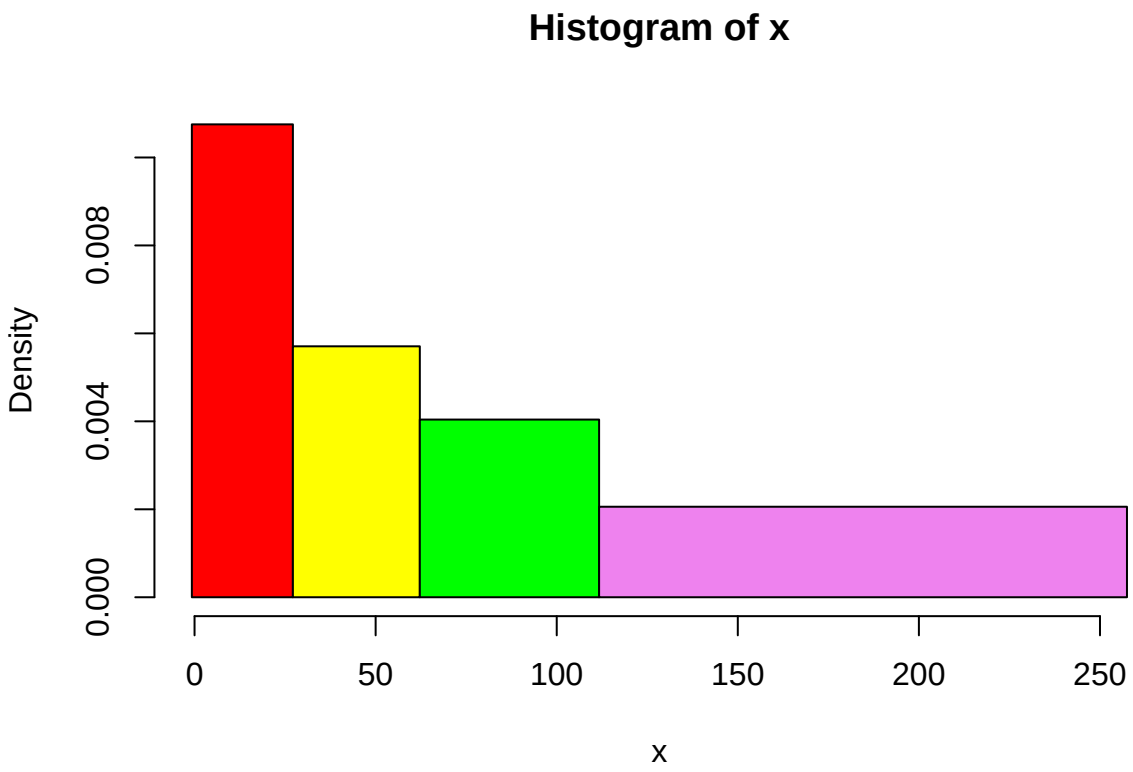
```
  hist(tab, prob=T, col=colors,breaks=seq(a0,ak,h))

}
```



La hauteur d'un rectangle est proportionnelle à l'effectif de sa classe.

Classe de même effectif

```
histoeff <- function(x, xlim=NULL, ...)
{
  sx <- sort(x)
  n <- length(x)
  k <- round(log(n)/log(2)+1)
  rangex <- max(x)-min(x)
  quantileVoulu <- quantile(x, seq(1,k-1)/k, max(x))

  breaks <- c(min(x)-0.025*rangex, quantileVoulu, max(x)+0.025*rangex)
  col <- 0
  if (is.null(xlim)) xlim<-c(breaks[1], breaks[k+1])
  colors = c("red", "yellow", "green", "violet", "orange", "blue", "pink", "cyan")

  hist(x, breaks=breaks, col=colors, xlim=xlim, probability=T)
}

histoeff(y)
```



Graphes de probabilités

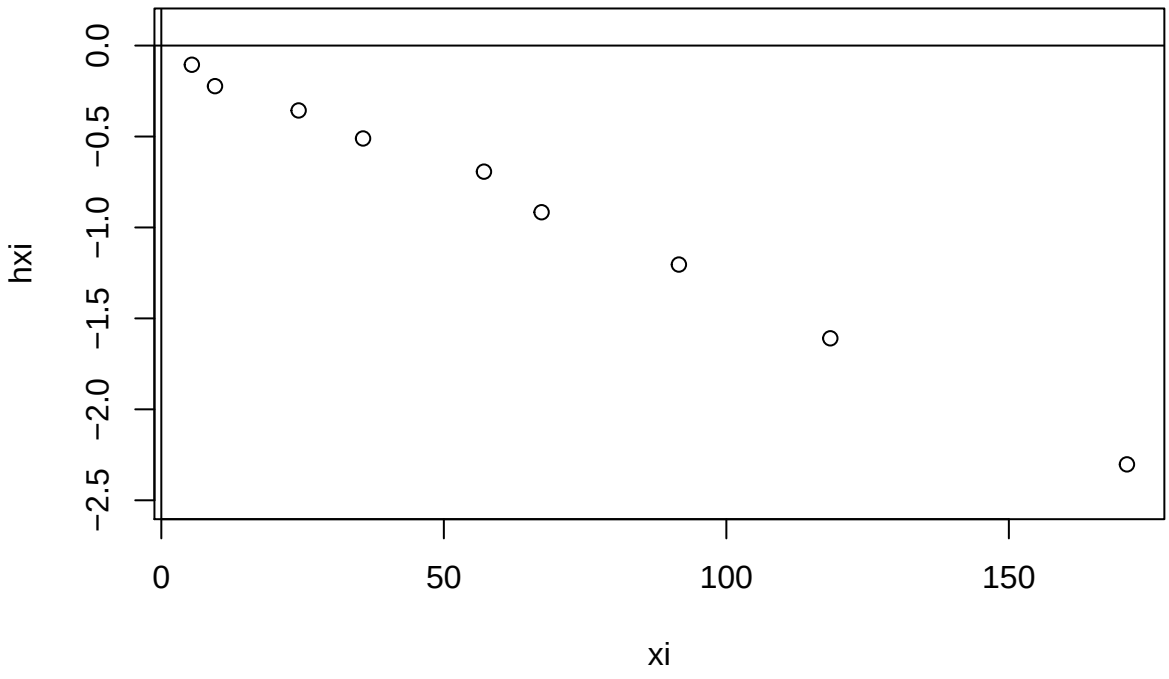
Si l'on a des observations dans un vecteur, on veut savoir la loi que suivent potentiellement ces observations. On pourrait se dire qu'il suffit de tracer la fonction de répartition empirique et de la comparer avec celles des différentes lois. Cependant, les fonctions de répartition sont très similaires entre elles, cela n'a pas d'intérêt. On peut alors transformer la fonction de répartition qui permettra de reconnaître visuellement la loi représentant les données.

Pour la loi exponentielle

Pour la loi exponentielle $exp(\lambda), F(x) = 1 - e^{-\lambda x}$, d'où $ln(1 - F(x)) = -\lambda x$. Le graphe de probabilité est donc le nuage de points $(x_i^*, ln(1 - \frac{i}{n})$ tel que $i \in 1, ..., n - 1$

```
graphProba <- function(xi,hxi,limy)
{
  plot(xi,hxi, ylim=limy)
  abline(v=0)
  abline(h=0)
}

# x représente le tableau non trié avec les observations sur les ampoules
# log(1 - seq(1:9)/10) est la transformation de la fonction de répartition de la loi exponentielle
# sort(x) [1:9] est le tableau x trié en ne gardant que les 9 premiers éléments
lastElem <- length(x)
blElem <- length(x)-1
```

```r
limy<-c(-2.5,0.1)
myxi <- sort(x)[1:blElem]
myhxi <- log(1-seq(1:blElem)/lastElem)

graphProba(myxi,myhxi,limy)
```



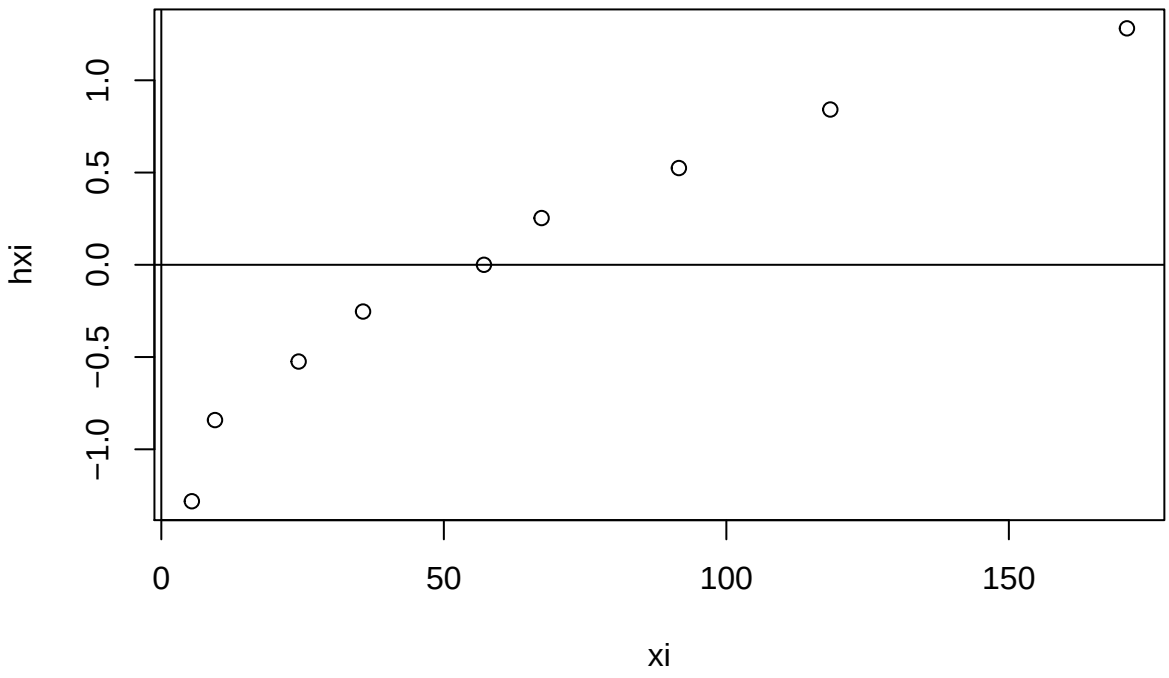
Les points sont très alignés, la durée de vie d'une ampoule semble suivre une loi exponentielle.

Pour la loi normale

On passe de la loi normale à la loi normale centrée réduite : Si X est de loi $\mathcal{N}(m, \sigma^2)$ alors $U = \frac{X-m}{\sigma}$ est de loi $\mathcal{N}(0,1)$ $F(x) = P(X \leq x) = P(U \leq \frac{x-m}{\sigma}) = \phi(\frac{x-m}{\sigma})$ où ϕ est la fonction de répartition de la loi normale centrée réduite. ϕ est strictement croissante donc inversible. On a alors : $\phi^{-1}(F(x)) = \frac{x-m}{\sigma}$

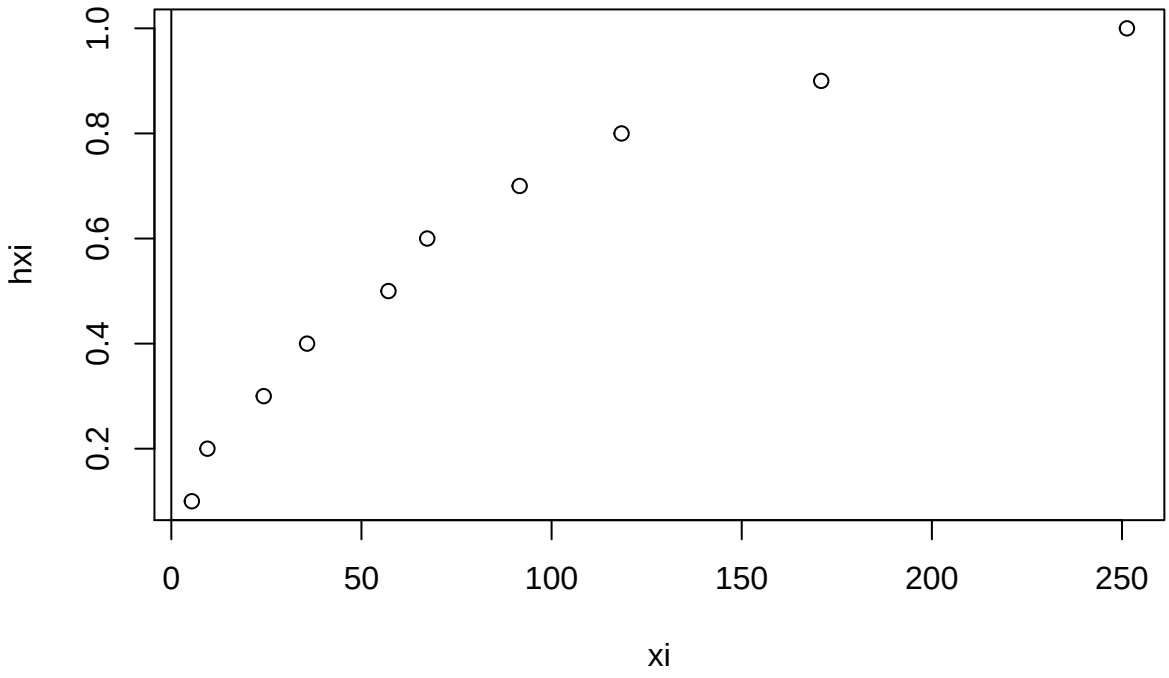
Le graphe de probabilité pour la loi noramel est le nuage de points $(x_i^*, \phi^{-1}(i/n))$ $\phi^{-1}(p)$ est donné par la fonction qpnorm(p)

```r
# qnorm(seq(1:9)/10) est la transformation de la fonction de répartition de la loi normale
lastElem <- length(x)
blElem <- length(x)-1
limy<-c()
myxi <- sort(x)[1:blElem]
myhxi <- qnorm(seq(1:blElem)/lastElem)

graphProba(myxi,myhxi,limy)
```



Pour la loi uniforme

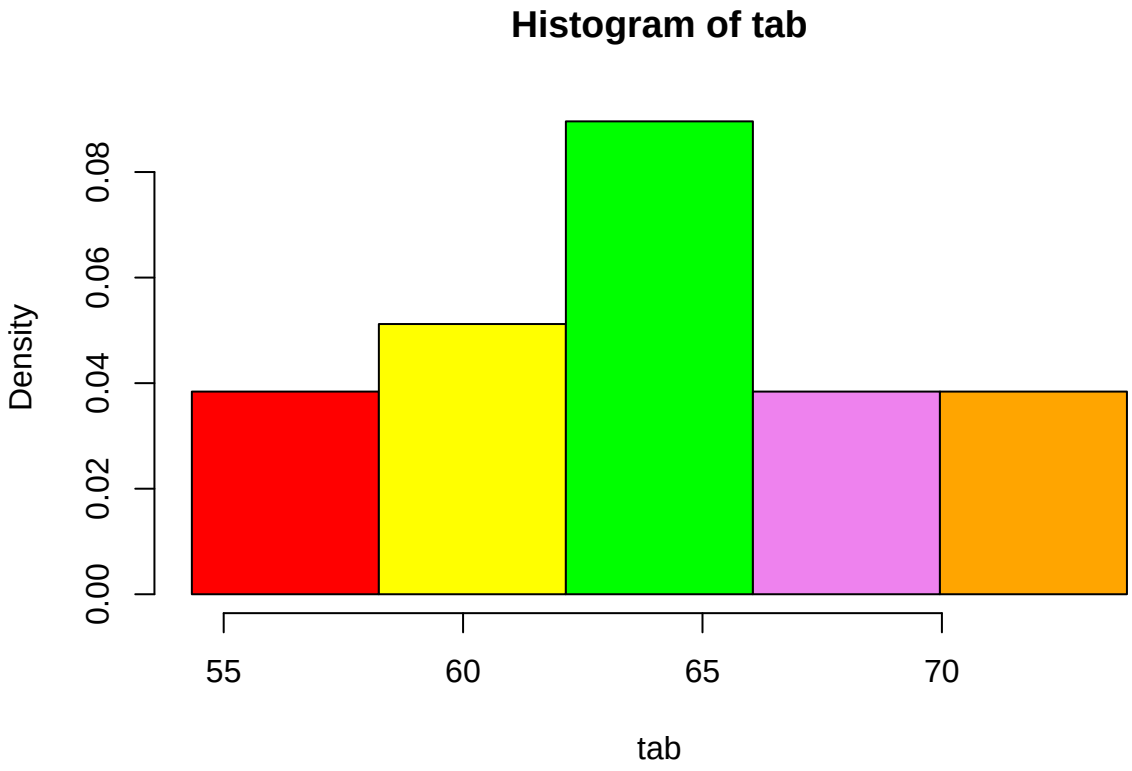
```
lastElem <- length(x)
blElem <- length(x)-1
limy<-c()
myxi <- sort(x)[1:lastElem]
myhxi <- seq(1:lastElem)/lastElem
graphProba(myxi, myhxi,limy)
```



Mise en oeuvre

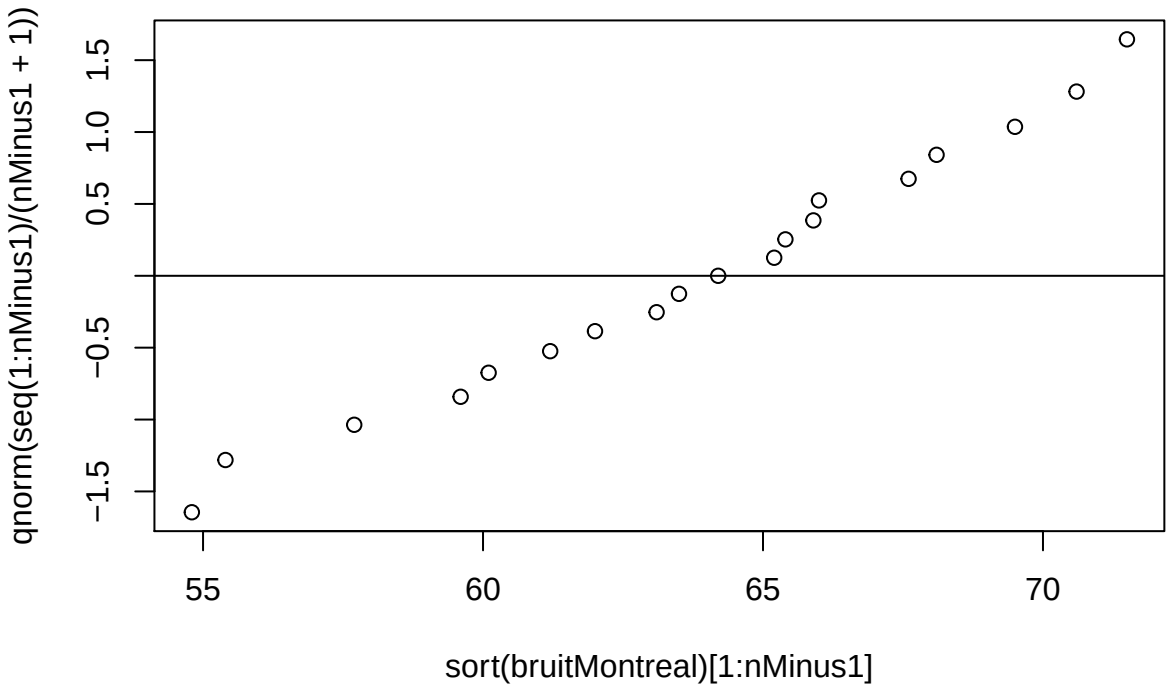
Je commence par créer un vecteur avec mes données, je le trie puis je fais l'histogramme pour exclure certaines hypothèses.

```
bruitMontreal <- c(54.8, 55.4, 57.7, 59.6, 60.1, 61.2, 62, 63.1, 63.5, 64.2, 65.2, 65.4,65.9,66,67.6,68

# Calcul du polygnone des fréquences
pfAbs <- c(0)
pfOrd <- c(0)
limAbs <- c(0)
limOrd <- 0.1

histogrammeClasse(bruitMontreal)
```



Cela ressemble à une loi normale. On peut se rapprocher de cette hypothèse en tracant le graphe de probabilité de la loi normale sur ces données.

```
# qnorm(seq(1:9)/10) est la transformation de la fonction de répartition de la loi normale
nMinus1 = length(bruitMontreal) - 1
plot(sort(bruitMontreal)[1:nMinus1], qnorm(seq(1:nMinus1)/(nMinus1+1)))
abline(h=0)
```



$\sqrt{x}$

Les points sont alignés, on peut donc donner une première approximation du niveau de bruit grâce à la fonction de répartition.

```
1 - pnorm(0.75)
```

```
## [1] 0.2266274
```

Loi de pareto

La loi de Pareto part du principe de pareto qui explique que 80% des effets proviennent de 20% des causes. En probabilité, si X suit une loi de pareto, alors : $P(X > x) = (\frac{x_m}{x})^k$ avec $x \geq x_m$

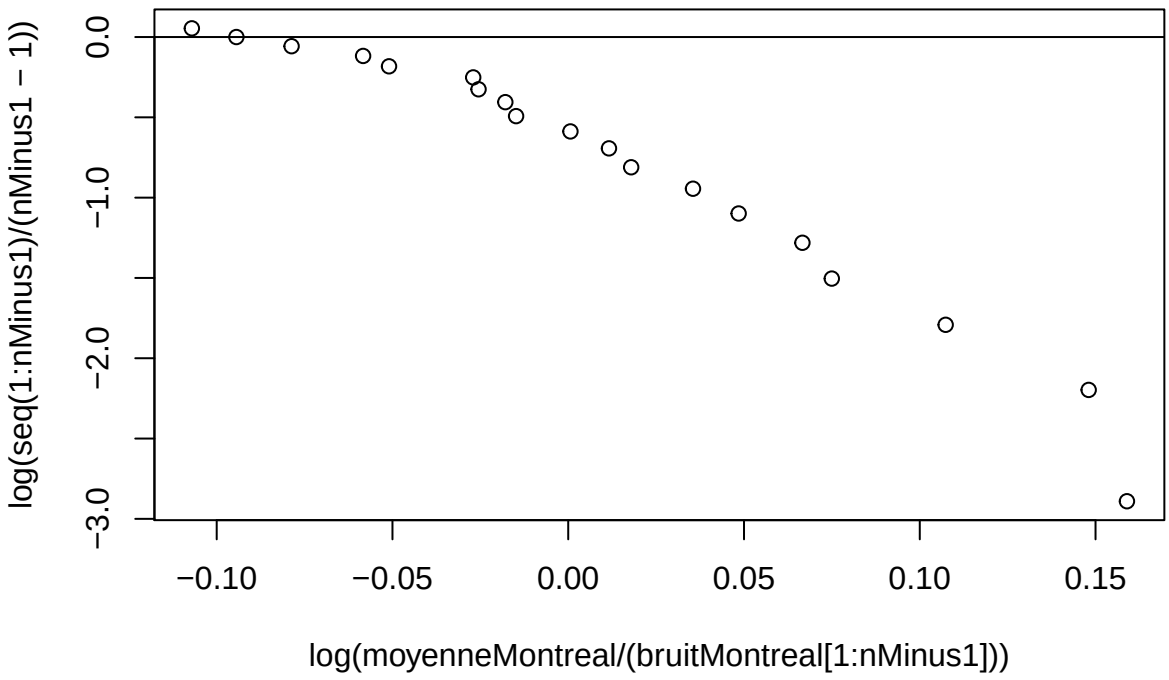
Un cas concret serai le service client d'une entreprise. Il suffirait de régler les problèmes de 20% des clients pour couvrir 80% des demandes.

Montrons qu'un jeu de données peut suivre une loi de Pareto. Il faut d'abord trouver le graphe de probabilité en transformant la fonction de répartition. En effet, on voit bien que la fonction dépend d'un paramètre k. $F(x) = 1 - (\frac{x_m}{x})^k$

$1 - F(x) = e^{kln(\frac{x_m}{x})}$

$ln(1 - F(x)) = kln(\frac{x_m}{x})$

```
moyenneMontreal <- mean(bruitMontreal)
plot(log(moyenneMontreal/(bruitMontreal[1:nMinus1])), log(seq(1:nMinus1)/(nMinus1-1)))
abline(h=0)
```



On peut voir que l'échantillon portant sur les prises de bruit à Montreal semble s'approcher d'une loi de Pareto.