

```

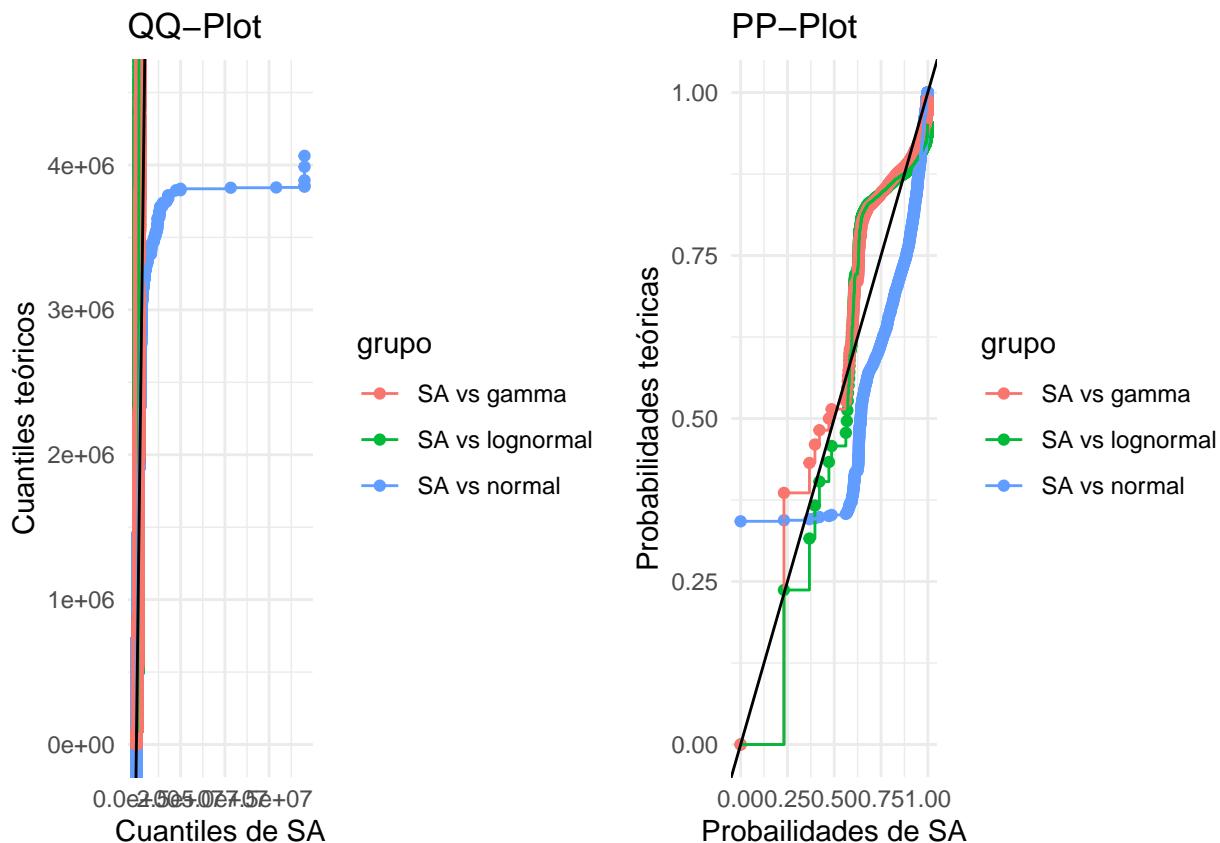
par(mfrow = c(1,1))
library(gridExtra)

##
## Adjuntando el paquete: 'gridExtra'

## The following object is masked from 'package:dplyr':
##     combine

grid.arrange(p, pp, ncol = 2)

```



## Prueba de estrés

La simulación adquiere las siguientes características

## Análisis de sensibilidad

```

#Sensibilidad
prueba = function(i,LMR,k){
  k = (1+i)*k
  z1 = k-LMR

```

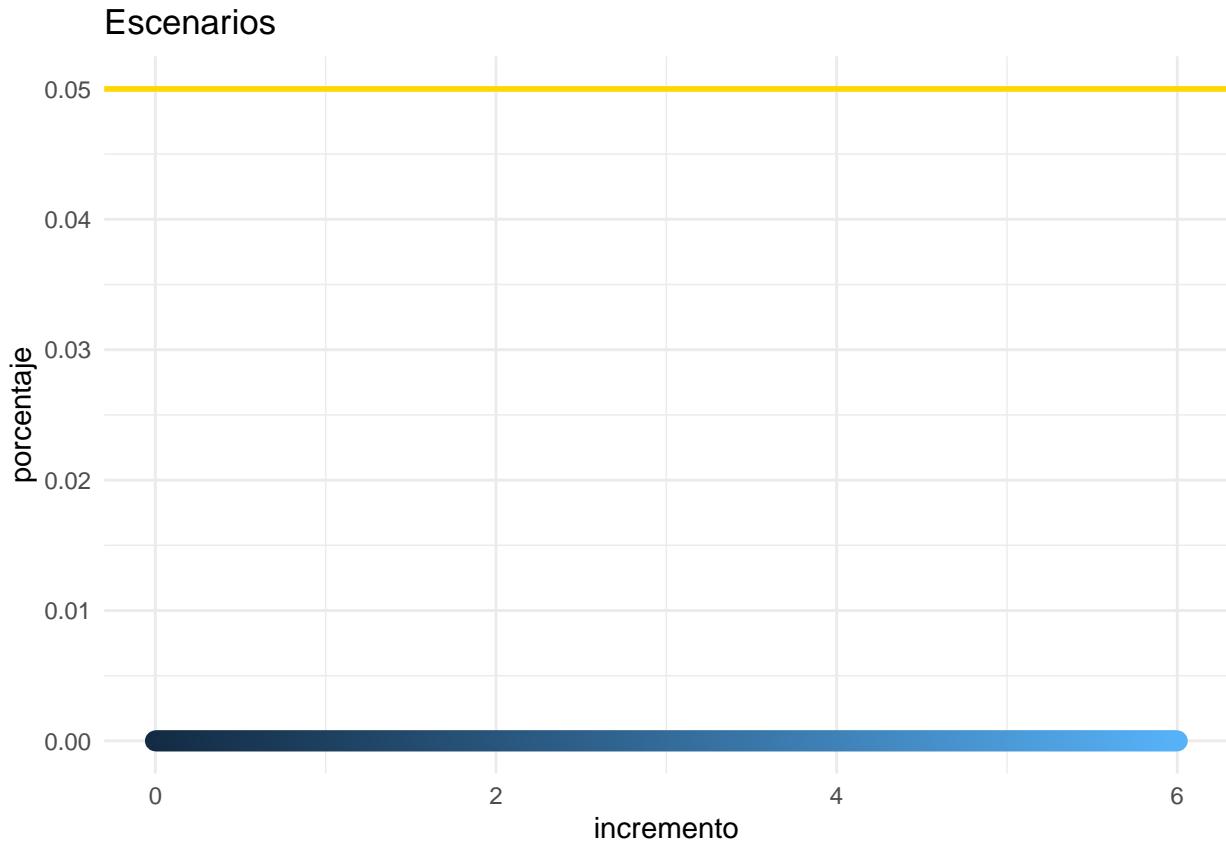
```

f = length(which(z1>0))/length(z1)
return(f)
}
library(ggplot2)
max(z)

## [1] 1644000

LMR = 12000000
d2 = seq ( from = 0, to = 6, by = 1/ 100)
escenarios = sapply (d2 , prueba , LMR = LMR, k=z )
# Datos de ejemplo
sensibilidad <- data.frame(
  incremento = d2,
  porcentaje = escenarios,
  categoria = c(1:length(escenarios))
)
# Scatter plot
ggplot(sensibilidad, aes(x = incremento, y = porcentaje, color = categoria)) +
  geom_point(size = 3) +
  labs(title = "Escenarios") +
  theme_minimal() + theme(legend.position = "none") +
  geom_hline(yintercept = .05, color = "gold", linetype = "solid", size = 1)

```



# Diversos Técnicos

## Preparación del modelado

En el siguiente código se aplica el tipo de cambio, y se construye la tabla P1 con los cuantiles de las sumas aseguradas.

Se ocupa el inciso 1 de la metodología basada en el algoritmo de simulación estocástica **Aceptación y Rechazo**

Las cuentas en dólares se convierten en dólares.

Se puede ver la estructura de los datos con los cuantiles. Los cuantiles permiten observar los datos a modelar ordenados y mostrando las frecuencias.

```
decil = seq(from = 0, to = 1, by = 1/10)
P1 = as.matrix(quantile(probs = decil, x = Te$SA), nrow = 11, ncol = 1)
Q = P1
Q = data.frame(Q)
Q$categoría = data.frame(categoría = c("1","2","3","4","5","6","7","8","9","10","11"))
Q$porcentaje = rep(.1, times = 11)
df = Q %>%
  group_by(categoría) %>%
  summarise(max(Q), Frecuencia = sum(porcentaje))
Q = cbind(as.numeric(df[1,2]), paste(as.character(as.numeric(df[1,2])), "-", as.character(as.numeric(df[2,2])), "-"))
for(j in 2:11){
  P = cbind(as.numeric(df[j,2]), paste(as.character(as.numeric(df[j,2])), "-", as.character(as.numeric(df[j,2])), "-"))
  Q = rbind(Q,P)
}
s = cbind(Q[-11,], df$Frecuencia[-1])
x = Te$SA
x = length(x)
s1 = as.numeric(s[,3])
s1 = s1*x
sum(s1) == x

## [1] TRUE

s = cbind(s,s1)
s

##                                     s1
## [1,] "25.21"      "25.21 - 1075000"    "0.1" "30050"
## [2,] "1075000"    "1075000 - 9.5e+07"   "0.1" "30050"
## [3,] "9.5e+07"    "9.5e+07 - 269.25"    "0.1" "30050"
## [4,] "269.25"     "269.25 - 3184.43"   "0.1" "30050"
## [5,] "3184.43"    "3184.43 - 5791.99"   "0.1" "30050"
## [6,] "5791.99"    "5791.99 - 13654.41"  "0.1" "30050"
## [7,] "13654.41"   "13654.41 - 24259.02" "0.1" "30050"
## [8,] "24259.02"   "24259.02 - 121498.93" "0.1" "30050"
## [9,] "121498.93"  "121498.93 - 573000"   "0.1" "30050"
## [10,] "573000"     "573000 - 794000"     "0.1" "30050"
```

```
## [,1]
## 0%    25.21
## 10%   269.25
## 20%   3184.43
## 30%   5791.99
## 40%   13654.41
## 50%   24259.02
## 60%   121498.93
## 70%   573000.00
## 80%   794000.00
## 90%   1075000.00
## 100%  95000000.00
```

Se actualiza la base para diversos técnicos.

### Ajuste de los datos

Dentro de la metodología se expondrá el inciso 2, **Aceptación y Rechazo**.

```
#Diversos tecnicos
#####inicio
library(MASS)
#Solo tomamos sumas aseguradas mayores que 0
C = C%>%filter(SA >0)
#En la variale M capturamos la suma asegurada
M = C$SA
#Se aplica una transformación Box Plot
M = log(M)-6.461
#Limpieza de outliers vía Box Plot
l = boxplot.stats(M)$stats
M= M[M >= l[1] & M <= l[5]]
```

### Ajuste estimador kernel

El kernel es el núcleo y consiste en una distribución con el mismo soporte que la distriución a modelar.

Primeramente se sabe por el teorema de lema de Glivenko-Cantelli que establece que la función de distribución empírica

$$F_n(x) = \frac{1}{n} \sum_{i=1}^n \chi_{(-\infty, x]}(x_i),$$

converge uniformemente en probabilidad a la distribución teórica propuesta, donde  $\{x_i\}_{i=1}^n$  es una muestra aleatoria de la variable aleatoria  $X$

$$F(x) = \mathbb{P}(X \leq x)$$

A pesar de ser una técnica usada ampliamente en las aplicaciones, tiene la limitante de que la distribución empírica no necesariamente es absolutamente continua, es decir, puede no tener función de densidad.

El método de aproximación Kernel pretende dada una muestra aleatoria  $\{x_i\}_{i=1}^n$ , definir una función  $\hat{f}_d(x)$  que satisface ser la densidad de

$$\hat{F}_n(x) = \int_{(-\infty, x)} \text{Kernel}\left(\frac{x - x_i}{h}\right) dF_n(x)$$

Al ser esta una convolución puede calcularse  $\int f * g = \int f \int g = \int f$  si  $g$  es función de densidad; en este caso si el kernel se escoge como una densidad digamos normal estandar, entonces esto se satisfacera y utilizando la media muestral entonces podrá estimarse la densidad buscada.

```
set.seed(1234)
#Kernel normal
cociente = function(x,h,M){
  return((x-M)/h)
}
kernel1= function(x,h,M){
  sumando = sum(dnorm(cociente(x,h,M),mean = 0, sd=1))
  return(sumando/(length(M)*h))
}
#La constante de escalamiento
h = bw.nrd0(M)
```

## Ajuste para encontrar función acotadora

Se define un cierto dominio para verificar que el Kernel usado es función de densidad. Se observa por inspección que la distribución Cauchy multiplicada por cierta constante acota por arriba a la densidad objetivo estimada con el método Kernel. Se usa la regla de Silverman ( $h$ ) para seleccionar el **Bandwidth**.

```
#Límites del eje x
a=min(M)
b=max(M)
dominio = seq(from = a, to = b, length.out= 1000)
#Densidad suavizada objetivo a modelar
#Aplicar el kernel al dominio
ya = sapply(dominio, FUN = kernel1,h=h,M=M)
#Densidad que acota por arriba a la densidad objetivo
#Aplicar densidad Cauchy al dominio
ye = sapply(dominio, FUN = dcauchy,location = 6, scale =5.9)
ye = 9*ye
```

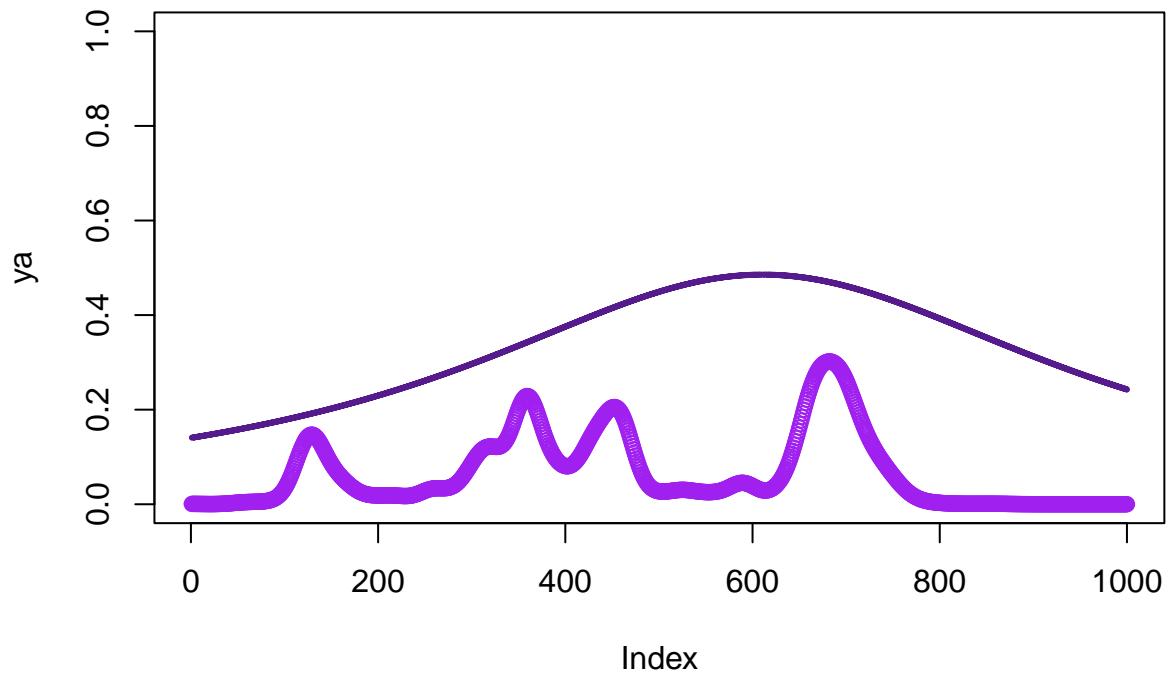
Se verifica que es una función de densidad

```
sum(ya)*diff(dominio)[1]
```

```
## [1] 0.9998075
```

Ahora se observa la aplicación del método de la función inversa conforme al inciso 2 de la metodología:

```
#curve(dlnorm(x,m,m1))
plot(ya, ylim=c(0,1), col="purple")
lines(ye,lwd = 3,type = "s", col = "purple4")
```



## Aplicación método de la función inversa

En aplicación del método de la función inversa pueden prepararse las variables insumo.

```
M = C$SA
M = log(M)-6.461
l = boxplot.stats(M)$stats
M= M[M >= l[1] & M <= l[5]]
h = bw.nrd0(M)
ter = h*length(M)
```

Definimos la función de densidad objetivo a modelar.

```
Objetivo = function(x,M1){
cociente =(x-M1)/h
  sumando = sum(dnorm(cociente,mean = 0, sd=1))
  return(sumando/(length(M1)*h))
}
```

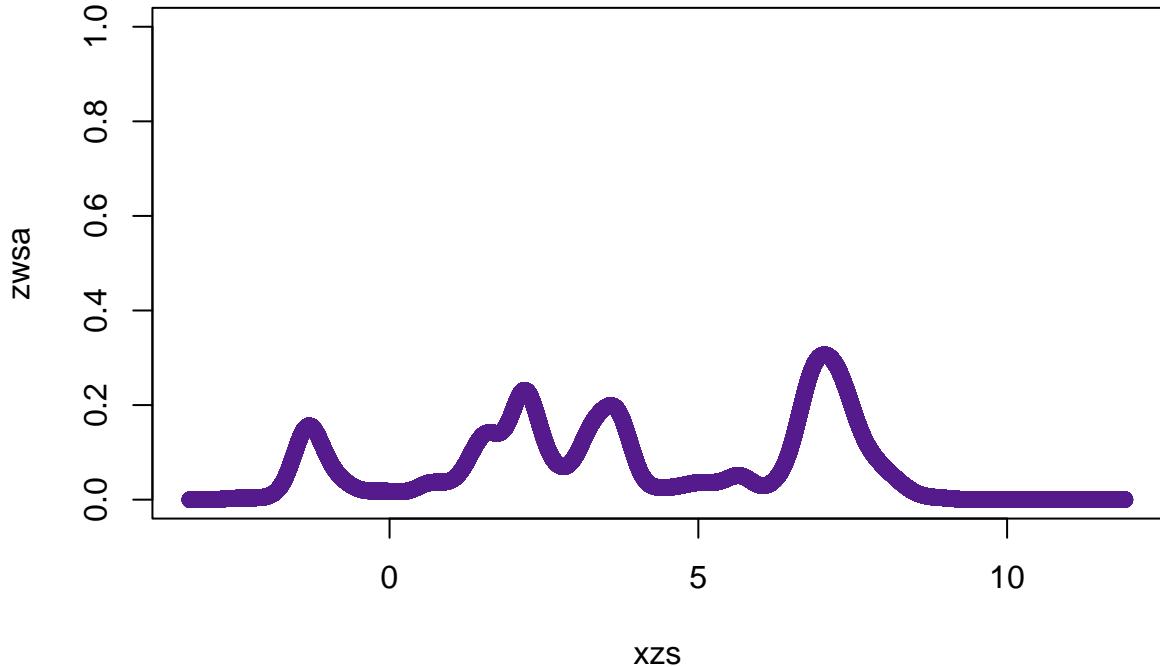
Se definen los parametros para observar gráficamente el comportamiento.

```
a = min(M)
b=max(M)
xzs = seq(from =a, to =b, length.out = 100000)
```

```

kwe =seq(from =1, to  =length(M), length.out = 2000)
M1 = M[kwe]
zwsa =sapply(xzs, Objetivo, M1)
plot(xzs,zwsa,ylim = c(0,1), col = "purple4")

```



Se generan números pseudoaleatorios Cauchy.

```

set.seed(1234)
longitud = 100000
u =c(1:longitud)
c = rcauchy(longitud,location = 6, scale =5.9)
c = c[c>0]
SA = list()

```

Una vez generados estos números pseudoaleatorios, se generan vectores con valuaciones  $U \sim \text{unif}(0, 9 \cdot f_{\text{cauchy}}(X))$ , aquí  $X \sim \text{Cauchy}$ .

```

for(i in 1:longitud){
  cota = dcauchy(c[i],location = 6, scale =5.9)
  cota = 9*cota
  u[i] = runif(1, 0, cota)
}

```

Ahora se generaran las corridas de la función objetivo  $\hat{f}_d(X)$

```

simulac = list()
M = C$SA
M = log(M)-6.461
l = boxplot.stats(M)$stats
M= M[M >= l[1] & M <= l[5]]
h = bw.nrd0(M)

```

Se generan las mallas de valores para valuar el Kernel.

```

kwe =seq(from =1, to =length(M), length.out = 2000)
M1 = M[kwe]
ter = h*length(M)

```

Se generan las valuaciones  $\hat{f}_d(X) = E[Kernel(\frac{X-M}{h})]$ .

```

for(i in 1:length(c)){
cociente = (c[i]-M1)/h
sumando = sum(dnorm(cociente,mean = 0, sd=1))
simulac[[i]] = sumando/ter
}
simulac = unlist(simulac)

```

Se aplica el algoritmo de la función inversa.

```

for(i in 1:length(c)){
  if(u[i] <= simulac[i]){
    SA[[i]] = c[i]
  }
}
SA = unlist(SA)

```

Se deshacen las transformaciones  $\log(M) - 6.461$  y se valida con bondad de ajuste.

```

Fer= C$SA
l = boxplot.stats(Fer)$stats
Fer= Fer[Fer >= l[1] & Fer <= l[5]]
z = exp(SA + 6.461)
ks.test(z,Fer)

```

```

## Warning in ks.test.default(z, Fer): p-value will be approximate in the presence
## of ties

##
##  Asymptotic two-sample Kolmogorov-Smirnov test
##
## data: z and Fer
## D = 0.14977, p-value = 0.2651
## alternative hypothesis: two-sided

```

Se valida ahora con datos transformados:

```

ks.test(M,SA)

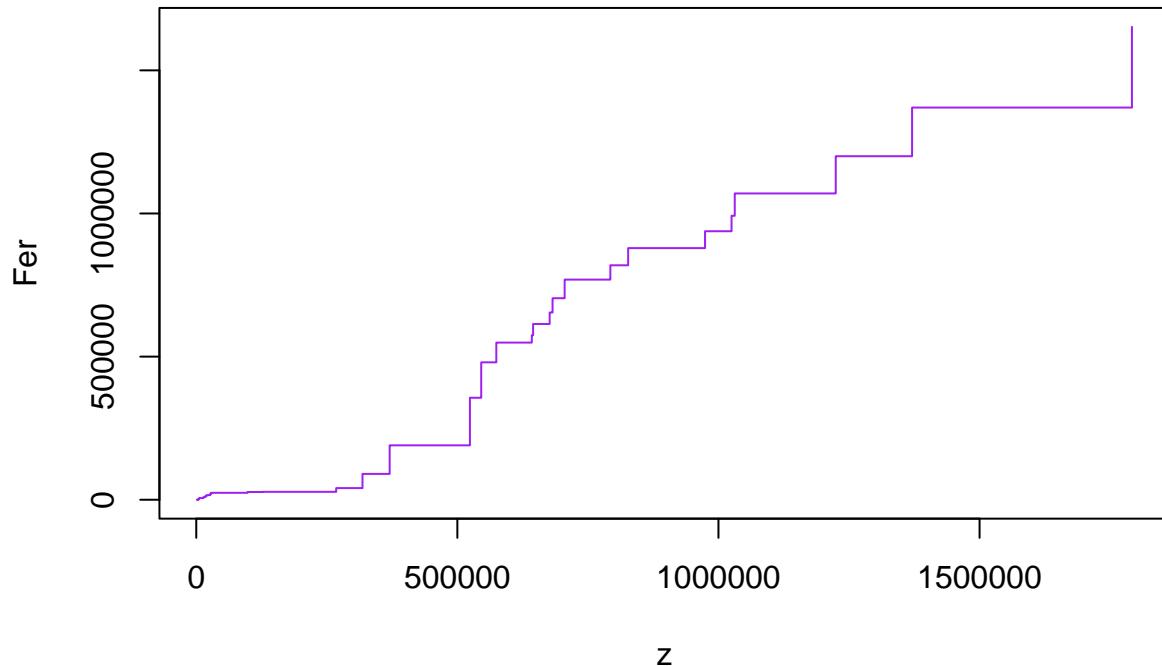
## Warning in ks.test.default(M, SA): p-value will be approximate in the presence
## of ties

##
##  Asymptotic two-sample Kolmogorov-Smirnov test
##
## data: M and SA
## D = 0.14219, p-value = 0.3229
## alternative hypothesis: two-sided

```

Se observa el gráfico cuantil-cuantil

```
qqplot(z,Fer, col = "purple", type = "s")
```



Observemos la estructura de la simulación

```

decil = seq(from = 0, to = 1, by = 1/10)
P11 = as.matrix(quantile(probs = decil, x = z), nrow = 11, ncol = 1)
Q11 = P11
Q11 = data.frame(Q11)

```

P1

```
## [,1]
## 0%    25.21
## 10%   269.25
## 20%  3184.43
## 30% 5791.99
## 40% 13654.41
## 50% 24259.02
## 60% 121498.93
## 70% 573000.00
## 80% 794000.00
## 90% 1075000.00
## 100% 95000000.00
```

P11

```
## [,1]
## 0%    1085.038
## 10%   3739.092
## 20%  4954.027
## 30% 7612.417
## 40% 15490.599
## 50% 27789.318
## 60% 288086.359
## 70% 568824.107
## 80% 686922.536
## 90% 1004754.893
## 100% 1791838.667
```

En gráficos

```
plot(ecdf(Te$SA), main = "CDF empírica y teórica", xlim = c(0, 686922.536))
lines(ecdf(z), col = "purple")
```