Pablo de la Fuente Parres Quintero

Procesos de renovación

Code

1.- Código que simule trayectorias de un proceso de renovación.

En este caso los tiempos interarribo se van a distribuir como una gamma de parámetros α y β.

Los inputs serán:

- tiempo final (t)
- parámetros de los tiempos interarribo (α y β)

El output será:

- vector con los tiempos de arribo (W_n)

Hide

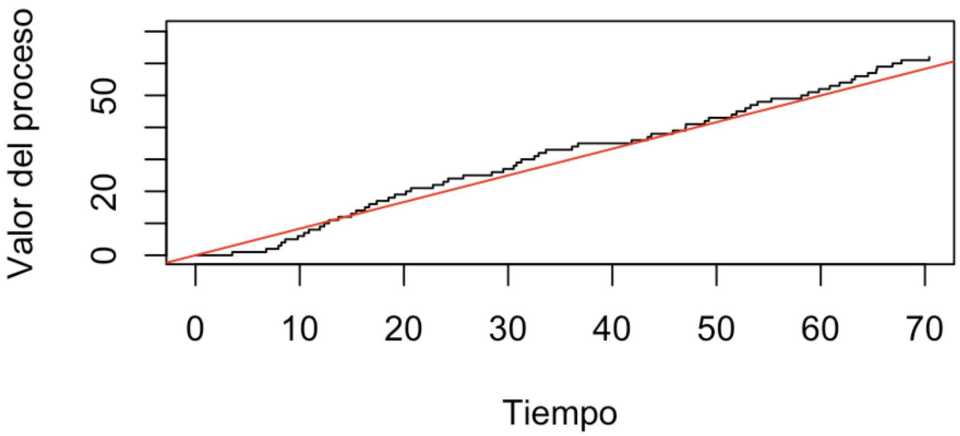
```
# Inputs
t <- 70
alpha <- 1.8
beta <- 1.5


# Output
W_n <- c(0) #el proceso siempre inicia en 0

# sumar tiempos interraribo hasta que se pase del tiempo final
while (W_n[length(W_n)] < t) {
  T_i <- rgamma(1, alpha, beta)
  W_n <- c(W_n, W_n[length(W_n)] + T_i)
}
```

Verificación: gráfica de una trayectoria y de la recta λt

Según el teorema de renovación, estas dos se deberán parecer mucho a tiempos grandes.

Hide

```
plot(W_n, 0:(length(W_n)-1), type = "s", xlim = c(0, t), ylim = c(0, max(W_n)), xlab = "Tiempo",ylab = "Valor del proceso")

abline(a = 0, b = 1/(alpha/beta), col = "red")
```



2.- Código que simule un proceso de renovación con premio.

Recordemos que un proceso de renovación con premio se ve como $X_t = \sum_{n=1}^{N_t} Y_n$.

En este caso,los tiempos interarribo son $T_n \sim \exp(\lambda)$ y los premios $Y_n \sim \Gamma(\alpha, \beta)$.

Los inputs serán:

- tiempo final (t)
- parámetro del tiempo interarribo (λ)
- parámetros de los premios (α y β)

Los outputs serán:

- vector con los tiempos de salto (W_N)
- vector con la suma de los saltos (X_t)

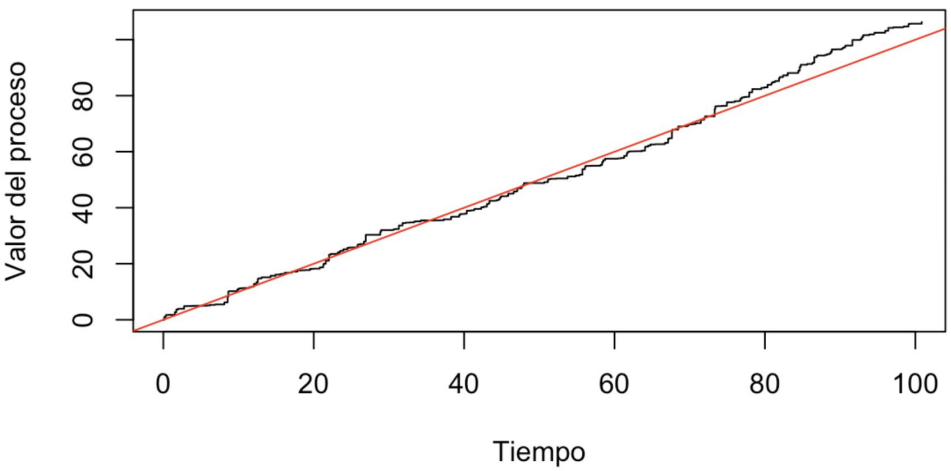
Hide

```
## Inputs
t <- 100
lambda <- 3
alpha <- 0.5
beta <- 1.5

## Outputs: vector con los tiempos de salto y vector con la suma de los saltos
W_n <- c(0)
X_t <- c(0)

# sumar tiempos interarribo hasta que sea mayor que t
# y en cada paso sumar un premio.
while (W_n[length(W_n)] < t) {
  T_i <- rexp(1, lambda)
  Y_i <- rgamma(1, alpha, beta)
  W_n <- c(W_n, W_n[length(W_n)] + T_i)
  X_t <- c(X_t, X_t[length(X_t)] + Y_i)
}
```

Para verificar el código, graficaremos el proceso junto con la recta $\frac{\mathbb{E}(Y_i)}{\mathbb{E}(T_i)}t$ y según el teorema de renovación se deberán parecer mucho.

Hide

```
# Gráfica del proceso
plot(W_n, X_t, type = "s", xlim = c(0, t), ylim = c(0, max(X_t)), xlab = "Tiempo", ylab = "Valor del proceso")

#Esperanza del proceso
abline(a = 0, b = (alpha/beta)/(1/lambda), col = "red")
```



3.- Cramer-Lundberg.

Recordemos que el proceso de Cramer-Lundberg se ve como $R_t = u + ct - \sum_{n=1}^{N_t} Y_n$ donde N_t es un proceso de Poisson.

Los inputs serán:

- tiempo final (t)
- capital inicial (u)
- prima (c)
- intensidad del Poisson (λ)
- parámetro de la distribución de los saltos (α)

Los outputs serán:

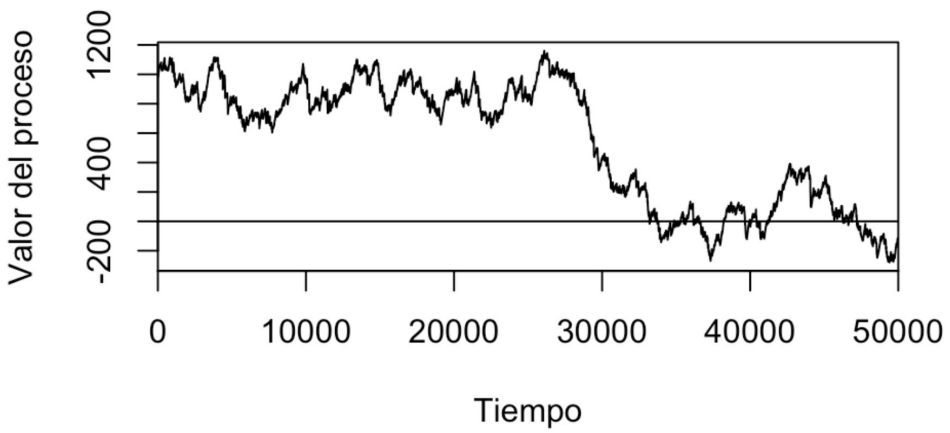
- vector con los tiempos de salto (W_n)
- vector con el valor del proceso (R_t)

Observación: como el proceso es creciente entre los saltos, para calcular la probabilidad de ruina, solo nos interesará cuánto vale en los momentos en que salta. Si se arruina, debe ser al momento de un salto y por eso, es suficiente con que nuestro vector solo tenga los R_{T_i}.

Por esta razón, la gráfica se verá diferente a como se ve normalmente.

Hide

```
## Inputs:
t <- 50000
u <- 1000
c <- 1
lambda <- 0.1
#Como aquí Yn ~ exp(alpha) entonces solo necesitamos un parámetro.
alpha <- 0.1005

## Simulación el proceso:
W_n <- c(0) #el tiempo inicia en 0.
R_t <- c(u) #el capital inicia en u.

while (W_n[length(W_n)] < t) {
  T_i <- rexp(1, lambda)
  W_n <- c(W_n, W_n[length(W_n)] + T_i)
  R_t <- c(R_t, c*T_i + R_t[length(R_t)] - rexp(1, alpha))
}

#Gráfica del proceso:
plot(W_n, R_t, type = "l", xlim = c(0, t), ylim = c(min(R_t), max(R_t)), xlab = "Tiempo", ylab = "Valor del proce
so", xaxs = "i")
abline(h = 0, xlim = c(0, t), ylim = c(min(R_t), max(R_t)))
```



Verificación

Para verificar el código verificaremos que la probabilidad de ruina coincide con la probabilidad teórica (usando Monte Carlo). Recordemos que la probabilidad teórica es $\psi(u) = \frac{\lambda}{\alpha c} \exp\{-(\alpha - \frac{\lambda}{c})u\}$.

Hide

```
iteraciones <- 100
procesos_arruinados <- 0
vector_probabilidades <- c(0)

#Monte Carlo. Repetiremos el proceso varias veces y contaremos en cuántas se arruina.
for(i in 1:iteraciones) {

  W_n <- c(0)
  R_t <- c(u)

  #Si el proceso se arruina lo detenemos.
  while(R_t[length(R_t)] > 0 & length(R_t) < t){
    T_i <- rexp(1, lambda)
    W_n <- c(W_n, W_n[length(W_n)] + T_i)
    R_t <- c(R_t, c*T_i + R_t[length(R_t)] - rexp(1, alpha))
  }

  #Contamos todos los procesos que se arruinaron.
  if(length(R_t) < t){
    procesos_arruinados <- procesos_arruinados + 1
  }

  #Este vector es para poder graficar.
  vector_probabilidades <- c(vector_probabilidades, (1/i)*procesos_arruinados)
}

#Calculamos la probabilidad de ruina.
probabilidad_ruina <- procesos_arruinados/iteraciones
probabilidad_ruina
```

```
[1] 0.6
```

Hide

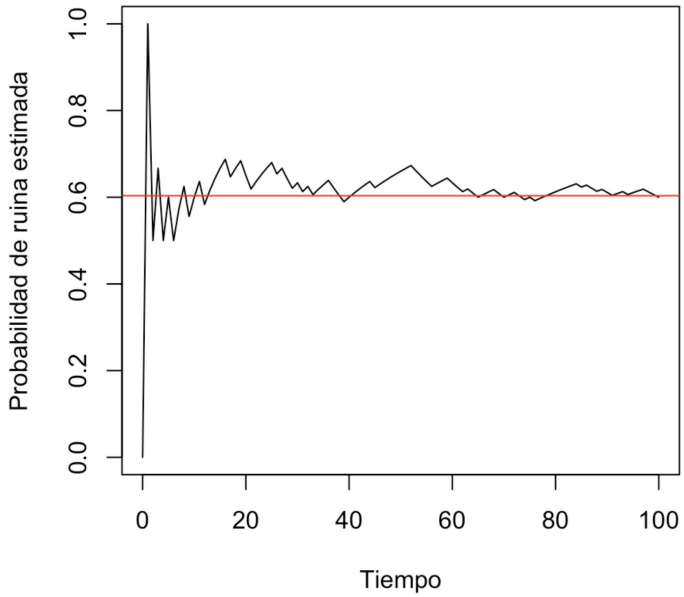
```
#Calculamos la probabilidad de ruina teórica.
psi <- (lambda/(alpha*c))*(exp(-(alpha-(lambda/c))*u))
psi
```

```
[1] 0.6035131
```

En la siguiente gráfica podemos ver cómo la probabilidad estimada se acerca a la teórica si hacemos más simulaciones. En rojo estará la probabilidad de ruina teórica.

Hide

```
plot(0:iteraciones, vector_probabilidades, xlab = "Tiempo", ylab = "Probabilidad de ruina estimada", type = "l")

abline(h = psi, col = "red")
```



Proceso de Cramer-Lundberg con un proceso de renovación general

Para calcular la probabilidad de ruina usaremos el mismo método pero sustituyendo el proceso Poisson por un proceso de renovación general con tiempos interrarribo $\mathrm{T_n} \sim \Gamma(10, 2)$ y con premios $\mathrm{Y_n} \sim \exp(\frac{1}{20})$.

Hide

```
iteraciones <- 100
procesos_arruinados <- 0
t <- 5000

u <- 1000
c <- 5


#Monte Carlo. Repetiremos el proceso varias veces y contaremos en cuántas se arruina.
for(i in 1:iteraciones) {

  W_n <- c(0)
  R_t <- c(u)

  #Si el proceso se arruina lo detenemos.
  while(R_t[length(R_t)] > 0 & length(R_t) < t){

    #tiempo interarribo
    T_i <- rgamma(1, 10, 2)
    W_n <- c(W_n, W_n[length(W_n)] + T_i)

    #restarle el premio
    R_t <- c(R_t, c*T_i + R_t[length(R_t)] - rexp(1, 1/20))
  }

  #Contamos todos los procesos que se arruinaron.
  if(length(R_t) < t){
    procesos_arruinados <- procesos_arruinados + 1
  }

}

#Calculamos la probabilidad de ruina.
probabilidad_ruina <- procesos_arruinados/iteraciones
probabilidad_ruina
```

```
[1] 0
```

Sí tiene sentido que la probabilidad de ruina sea 0.

Por el teorema de renovación con premio, sabemos que a tiempos grandes se cumple que

$$\sum_{n=1}^{N_t} Y_n \approx \frac{\mathbb{E}(Y_n)}{\mathbb{E}(T_n)} = \frac{5}{20} = \frac{1}{4}.$$

Por lo tanto, podemos estimar que a tiempos grandes, nuestro proceso de Cramer-Lundberg

$$R_t = u + ct - \sum_{n=1}^{N_t} Y_n \approx 1000 + 5t - \frac{1}{4}t$$

$$\Rightarrow R_t \approx 1000 + \frac{19}{4}t.$$

Es decir, a tiempos grandes nuestro proceso se ve como una recta con pendiente positiva y por eso esperaríamos que nunca se arruine.

Podemos graficar una trayectoria para confirmarlo.

Hide

```r
t <- 500
u <- 1000
c <- 5

W_n <- c(0)
R_t <- c(u)

while(length(R_t) < t){

    #tiempo interarribo
    T_i <- rgamma(1, 10, 2)
    W_n <- c(W_n, W_n[length(W_n)] + T_i)

    #restarle el premio
    R_t <- c(R_t, c*T_i + R_t[length(R_t)] - rexp(1, 1/20))
}

plot(1:length(R_t), R_t, xlim = c(0, t), xlab = "Tiempo", ylab = "Valor del proceso", type = "l", xaxs = "i")
abline(a = 1000, b = (19/4), col = "red")
```