Simulación

Fabiola Vázquez

---

1. Generación de tiempos de llegada

Queremos generar los tiempos de llegadas de un proceso Poisson con razón λ, en los casos en que dicho proceso sea homogéneo o no. Haremos uso del lenguaje R [1] en el IDE **R Studio** [3]. Utilizaremos el siguiente algoritmo [2]:

Paso 1: $t = 0, I = 0$.

Paso 2: Generar un número aleatorio U.

Paso 3: $t = t - \frac{1}{\lambda} \log U$. Si $t > T$, terminar.

Paso 4: $I = I + 1, S(I) = t$.

Paso 5: Ir al Paso 2.

En el algoritmo, `t` es el tiempo, `I` es el número de eventos que han transcurrido hasta el tiempo `t`, y `S(t)` es el tiempo del evento más reciente.

En este proceso es importante crear números pseudoaleatorios, como lo menciona el Paso 2 de nuestro algoritmo, para ello hacemos uso de la función `runif`. El siguiente código muestra la creación de la función `llegadasPH` para generar los tiempos de llegada.

```
llegadasPH <- function(lambda, T) {
  S <- vector("numeric")
  t <- 0

  while(t <= T){
      U <- runif(1, 0, 1)
      t <- t - (1/lambda)*log(U)
      if(t > T){
          return(S)
      } else{
          S <- append(S, t)
      }

  }
}
```

La función depende de los parámetros `lambda` y `T` que son nuestra tasa de llegada y el tiempo a simular, respectivamente. Ya cargada la función, se le asigna valores a dichos parámetros y se generan los tiempos de llegada. Para poder reproducir los resultados es necesario fijar una semilla.

```
>set.seed(10)
>llegadasPH(0.5,20)
[1]  1.356603  3.719927  5.422302  6.155458 11.082469 14.061902
[7] 16.647287 19.248952
```

Hacemos algo similar para el proceso de Poisson no homogéneo. Creamos una función llamada `llegadasPNH`.

```
#De momento, tomaremos lambda(t) = 2t+1 solo para ejemplificar
#Esto hace que lambda = 2T+1 por ser creciente
llegadasPNH <- function(T){
        lambda = 2*T + 1
        S <- list()
        t <- 0

        while(t <= T){
                U <- runif(1,0,1)
                t <- t - (1/lambda)*log(U)
                if(t > T){
                  return(S)
                } else{
                        U <- runif(1,0,1)
                        if(U <= (2*t + 1)/lambda){
                                S <- append(S, list(tiempo=t, tipo="llegada"))
                        }
                }
        }
}
```

Fijamos semilla y asignamos valores a los parámetros `lambda` y `T`

```
> set.seed(10)
> llegadasPH(0.5,20)
[1]  1.356603  3.719927  5.422302  6.155458 11.082469 14.061902
[7] 16.647287 19.248952
```

2. Simulación de línea de espera con un servidor

Ahora, consideremos un sistema al cual los clientes llegan de acuerdo a un proceso de Poisson no homogéno, tenemos solo un servidor, si este esta desocupado el cliente pasa al servicio, en caso contrario tiene que hacer fila. Cuando el servidor termina con un cliente, el siguiente en pasar es el que ha estado esperando más tiempo (disciplina **FIFO**). En el siguiente código, hecho en python [4], `n_a` es el número de llegadas, `n_d` es el número de salidas y `n` es el número de clientes en el sistema.

```
import random, math
def llegadasPoissonNoHomo(T):
```

```
    Lambda = 2*T + 1
    S = []
    t = 0
    while(t <= T):
        U = random.uniform(0,1)
        t = t - (1/Lambda)*math.log(U)
        if t > T:
            return S
        else:
            U = random.uniform(0,1)
            if U <= (2*t + 1)/Lambda:
                S.append((t, "llegada"))

def llegadasPoisson(T, Lambda):
    S = []
    t = 0
    while t <= T:
        U = random.uniform(0,1)
        t = t - (1/Lambda)*math.log(U)
        if t > T:
            return S
        else:
            S.append((t, "llegada"))

def tiempoServicio(mu):
    U = random.uniform(0,1)
    return -math.log(1-U)/mu

def procesaLlegada(evento):
    global t, n, n_a, n_q, servidor_ocupado, mu
    t = evento[0]
    n = n+1
    n_a = n_a + 1

    if servidor_ocupado == True:
        n_q = n_q + 1
    else:
        servidor_ocupado = True
        salida = (evento[0] + tiempoServicio(mu), "salida")
        eventos.append(salida)

def procesaSalida(evento):
    global t, n, n_s, n_q, servidor_ocupado
    t = evento[0]
    n = n-1
    n_s = n_s + 1
    if n_q > 0:
```

```
        n_q = n_q - 1
        salida = (evento[0] + tiempoServicio(mu), "salida")
        eventos.append(salida)
    else:
        servidor_ocupado = False

n, n_a, n_s, n_q, t = 0, 0, 0, 0, 0
servidor_ocupado = False
mu = 1

eventos = llegadasPoissonNoHomo(2)

print("Al tiempo " + str(t) + " tenemos: ")
print(str(n) + " personas en el sistema; ")
print(str(n_a) + " llegadas;")
print(str(n_s) + " salidas;")
print(str(n_q) + " gente en fila. \n----------\n")

while(eventos):
    eventos.sort()

    siguiente_evento = eventos.pop(0)

    if siguiente_evento[1] == 'llegada':
        procesaLlegada(siguiente_evento)

    if siguiente_evento[1] == 'salida':
        procesaSalida(siguiente_evento)

    print("Al tiempo " + str(t) + " tenemos: ")
    print(str(n) + " personas en el sistema; ")
    print(str(n_a) + " llegadas;")
    print(str(n_s) + " salidas;")
    print(str(n_q) + " gente en fila. \n----------\n")
```

Referencias

[1] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2020.

[2] Sheldon M. Ross. *Simulation*. Pearson, Berkeley, California, 1999.

[3] RStudio Team. *RStudio: Integrated Development Environment for R*. RStudio, PBC., Boston, MA, 2020.

[4] Guido van Rossum. *Python*. Python Software Foundation, Países Bajos, 2020.