

# Trabajo de Simulación de Distribuciones de Probabilidad

Francisco Javier Jiménez Montes

Juan Marqués Garrido

Pablo Jesús Moreno Polo

# Introducción

El propósito de esta memoria es describir el algoritmo programado para realizar nuestra simulación de la distribución de Poisson de variable  $\lambda$  con muestra de tamaño  $N$ .

El lenguaje usado para programar esta simulación es R.

En esta memoria se hará referencia constante a estas tres variables:

```
size <- 100000
lambda <- 32
pois_vals <- integer()
```

Donde *size* es el tamaño de nuestra muestra, *lambda* es la media de nuestra distribución de *Poisson* y *pois\_vals* nuestro array con los valores de nuestra muestra.

## Generación de la Muestra

Para generar nuestra muestra de valores necesitamos generar valores  $U_i \in (0,1)$ . Necesitamos obtener los valores necesarios aplicando  $V_i = -\ln(U_i)$ . Con estos  $V_i$  vamos a obtener los valores de nuestra variable aleatoria  $X \sim P(\lambda)$ , la cual es el numero de términos  $V_i$  que se han sumado antes de que la suma exceda  $\lambda$ .

La función que ha sido programada para generar los valores de la variable  $X$  con los parámetros  $N$  y  $\lambda$  es:

```
gen_poisson_values <- function(lambda) {
  sum <- 0
  count <- 0
  while (sum < lambda) {
    sum <- sum - log(runif(1))
    count <- count + 1
  }
  return(count - 1)
}
```

Para calcular nuestros  $N$  y ademas obtener el menor y el mayor valor obtenidos (los cuales nos serán útiles posteriormente para generar nuestros histogramas), tenemos estas tres lineas:

```
for (i in 1:size) {
  pois_vals <- append(pois_vals, gen_poisson_values(lambda))
}
min_val <- min(pois_vals)
max_val <- max(pois_vals)
```

Donde *size* es nuestra  $N$  y *pois\_vals* es un *array* en el cual almacenamos nuestros valores.

`min_val` y `max_val` son el menor y el mayor valor obtenido en nuestra distribución de valores respectivamente, y sus valores son:

```
> min_val  
[1] 12  
> max_val  
[1] 61
```

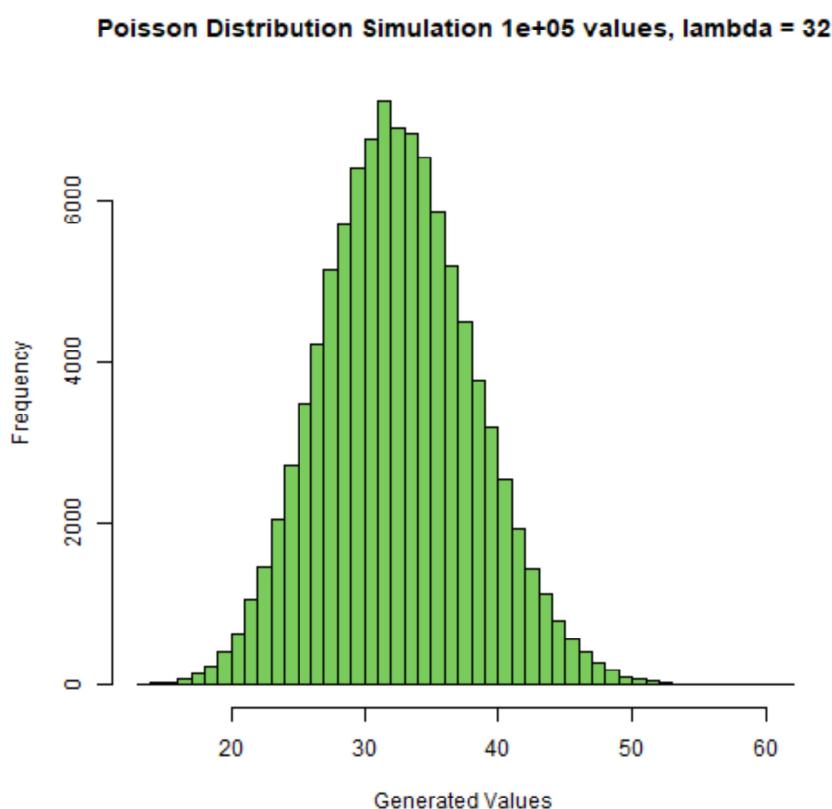
## Representando nuestros datos con un histograma

Ahora tenemos nuestro vector con valores, pero visualizar estos datos es bastante complicado con un vector; es por eso que aquí entra en juego representar nuestros datos de una forma gráfica en un histograma.

Para generar un histograma usamos la función `hist()` con estos parámetros.

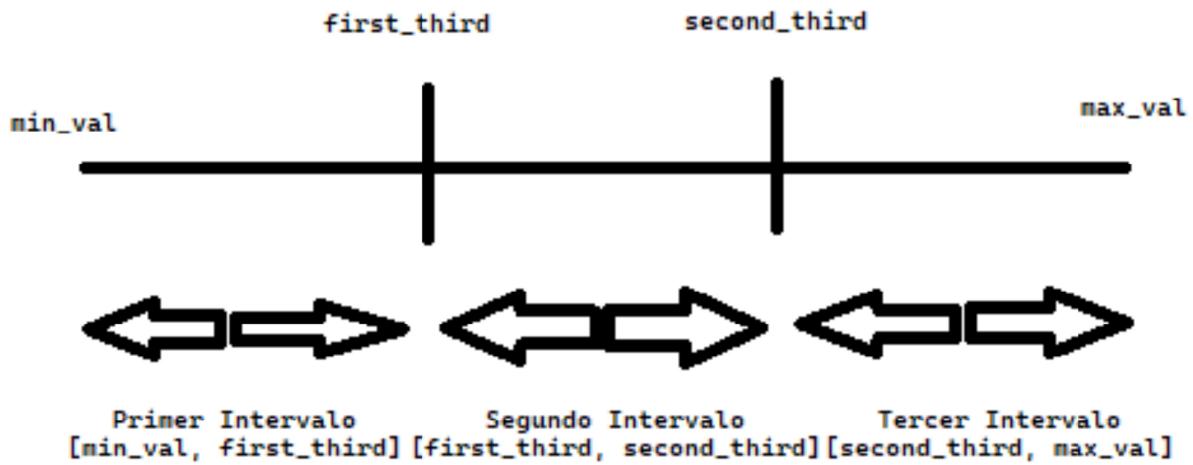
```
hist(pois_vals,  
     main = paste("Poisson Distribution Simulation",  
                 size, "values, lambda =", lambda  
                 ),  
     xlab = "Generated Values",  
     xlim = c(min_val, max_val),  
     breaks = max_val - min_val,  
     col = "#79cc5b"  
)
```

Si ejecutamos esta línea obtenemos el siguiente histograma:



## Estimación de probabilidad de intervalos de extremos finitos

Hemos decidido estimar la probabilidad en tres intervalos, definidos en la siguiente imagen:



Para ello hemos definido 3 funciones, la primera función es usada para calcular el intervalo con los valores de la distribución de *Poisson*, proporcionados por la función `ppois`.

```
calc_ppois_interval <- function(inferior, superior, lambda) {  
  return(ppois(superior, lambda) - ppois(inferior, lambda))  
}
```

La segunda función es usada para calcular la probabilidad del intervalo con los valores que hemos generado y almacenado

```
calc_array_interval <- function(inferior, superior, array_vals, size) {  
  return((length(which(pois_vals < superior)) -  
         length(which(pois_vals < inferior))) / size)  
}
```

Estas dos funciones son usadas en nuestra tercera función la cual se encarga de llamar a ambas funciones e imprimir por pantalla sus resultados y el error cometido entre el valor teórico y el valor experimental.

```
calc_interval <- function(inferior, superior, lambda, array_vals, size) {  
  prob_dist <- calc_ppois_interval(inferior, superior, lambda)  
  prob_sample <- calc_array_interval(inferior, superior, array_vals, size)  
  error <- abs(prob_dist - prob_sample)  
  message(sprintf("Interval: [%f, %f]", inferior, superior))  
  message(sprintf("\t- Ppois Function Values -> %f", prob_dist))  
  message(sprintf("\t- Generated Values -> %f", prob_sample))  
  message(sprintf("\t- Error -> %f", error))  
}
```

Antes de aplicar nuestras tres funciones necesitamos calcular la diferencia entre el valor mas alto y el valor mas bajo de nuestra distribución; posteriormente lo dividimos entre tres y lo usamos para obtener el *first\_third* y el *second\_third*.

El código para estas tres variables es el siguiente

```
difference <- (max_val - min_val) / 3
first_third <- min_val + difference
second_third <- min_val + 2 * difference
```

Con estos tres valores aplicamos las funciones definidas anteriormente para calcular nuestros tres intervalos.

```
## [min_val, first_third]
calc_interval(min_val, first_third, lambda, pois_vals, size)
## [first_third, second_third]
calc_interval(first_third, second_third, lambda, pois_vals, size)
## [second_third, max_val]
calc_interval(second_third, max_val, lambda, pois_vals, size)
```

La salida por pantalla es la siguiente:

```
> ## [min_val, first_third]
> calc_interval(min_val, first_third, lambda, pois_vals, size)
Interval: [12.000000, 28.333333]
  - Ppois Function Values -> 0.274075
  - Generated Values -> 0.273770
  - Error -> 0.000305
> ## [first_third, second_third]
> calc_interval(first_third, second_third, lambda, pois_vals, size)
Interval: [28.333333, 44.666667]
  - Ppois Function Values -> 0.708581
  - Generated Values -> 0.709230
  - Error -> 0.000649
> ## [second_third, max_val]
> calc_interval(second_third, max_val, lambda, pois_vals, size)
Interval: [44.666667, 61.000000]
  - Ppois Function Values -> 0.017295
  - Generated Values -> 0.016990
  - Error -> 0.000305
```

## Estimaciones puntuales de la media y de la varianza muestral

La estimación de dichos estadísticos es sencilla de realizar, pues se puede hacer en tan solo 2 líneas de código.

```
pois_mean <- mean(pois_vals)
pois_var <- var(pois_vals)
pois_mean
pois_var
```

Los valores obtenidos son los siguientes:

```
> pois_mean
[1] 31.99686
> pois_var
[1] 31.86269
```

## Aplicación del Teorema Central del Límite

El Teorema Central del Límite, también llamado CLT (o *Central Limit Theorem*), es aplicado sobre una muestra  $\{x_1, x_2, \dots, x_n\}$  para generar una nueva muestra  $\{y_1, y_2, \dots, y_n\}$ . Cada valor  $y_i$  se calcula de la siguiente forma:

$$y_i = \frac{x_i - \bar{x}}{\sqrt{s^2}}$$

La función CLT se define en R de la siguiente forma:

```
clt_function <- function(x) {
  num_aux <- (x - pois_mean) / sqrt(pois_var)
  return(num_aux)
}
```

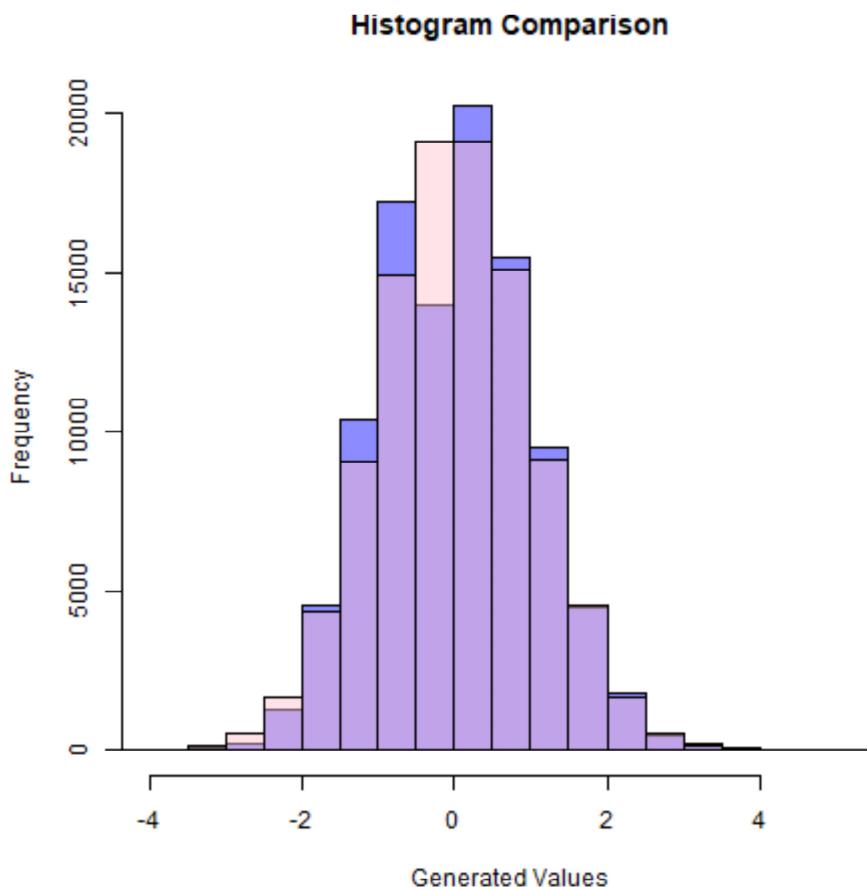
Posteriormente, aplicamos la función sobre nuestro vector de valores y obtenemos su media y desviación típica muestral.

```
clt_values <- sapply(pois_vals, clt_function)
clt_mean <- mean(clt_values)
clt_sd <- sd(clt_values)
```

Comprobando ambos valores observamos que coinciden con los resultados esperados.

```
> clt_mean
[1] 3.258299e-16
> clt_sd
[1] 1
```

Si superponemos un histograma con valores generados aleatoriamente de una distribución normal con un histograma de nuestros valores tras aplicarle la función CLT obtenemos la siguiente imagen.



Donde el histograma azul representa los valores después de aplicar la función CLT y el rosado representa los valores generados de la distribución normal.