INICIO: ABRIL 2018

CURSO VIRTUAL

PROGRAMACIÓN ESTADÍSTICA CON R

UNIVERSIDAD PERUANA
CAYETANO HEREDIA
Escuela de Posgrado

Programación Estadística con R

UPCH

Abril 2018

UNIVERSIDAD PERUANA
CAYETANO HEREDIA
ESCUELA DE POSGRADO

Programación en R

1. Funciones de Distribución en R

UNIVERSIDAD PERUANA
CAYETANO HEREDIA
Escuela de Posgrado

Método congruencial multiplicativo : Ejemplo 1

Generar 50 números pseudoaleatorios a partir del generador congruencial multiplicativo.

$$x_n = 171x_{n-1}(mod\ 30269)$$

$$u_n = x_n/30269$$

con semilla 27218

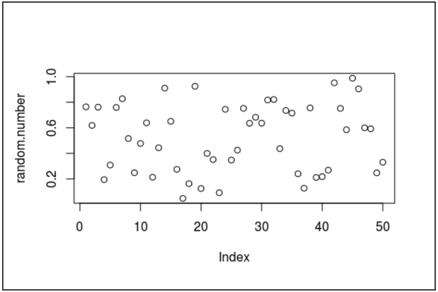
GENERACIÓN DE NÚMEROS PSEUDOALEATORIOS

Método congruencial multiplicativo : Ejemplo 1

```
random.number<-numeric(50) # vector vacio de 50 elementos
random.seed<-27218 # semilla
for (j in 1:50){
# construimos el vector random.number elemento a elemento
random.seed<-(171*random.seed) %% 30269
random.number[j]<-random.seed/30269
}
random.number
```

Método congruencial multiplicativo : Ejemplo 1

```
plot(random.number)
```

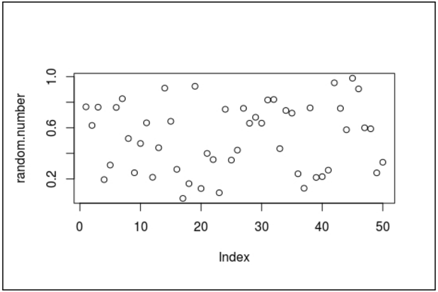


Método congruencial multiplicativo (Otra forma) : Ejemplo 1

```
x<-numeric(50)
semilla<-27218
x[1]=(171*semilla) %% 30269
for(i in 2:50){x[i]=(171*x[i-1]) %% 30269}
NumerosAleatorios<-x/30269
NumerosAleatorios
```

Método congruencial multiplicativo (Otra forma) : Ejemplo 1

```
plot(NumerosAleatorios)
```



UNIVERSIDAD PERUANA CAYETANO HEREDIA
ESCUELA DE POSGRADO

Método congruencial multiplicativo : Ejemplo 2

Generar 50 números pseudoaleatorios a partir del generador congruencial

$$x_n = 69069x_{n-1}(mod 2^{37})$$

$$u_n = x_n/(2^{37})$$

UNIVERSIDAD PERUANA CAYETANO HEREDIA
ESCUELA DE POSGRADO

Método congruencial multiplicativo : Ejemplo 2

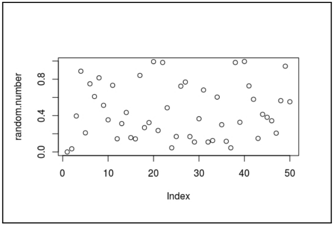
```
random.number<-numeric(50)
random.seed<-1
for (j in 1:50)
{random.seed<-(69069*random.seed) %% (2^(37))
random.number(j)<-random.seed/(2^(37))
}
random.number
```

UNIVERSIDAD PERUANA
CAYETANO HEREDIA
Escuela de Posgrado

GENERACIÓN DE NÚMEROS PSEUDOALEATORIOS

Método congruencial multiplicativo : Ejemplo 2

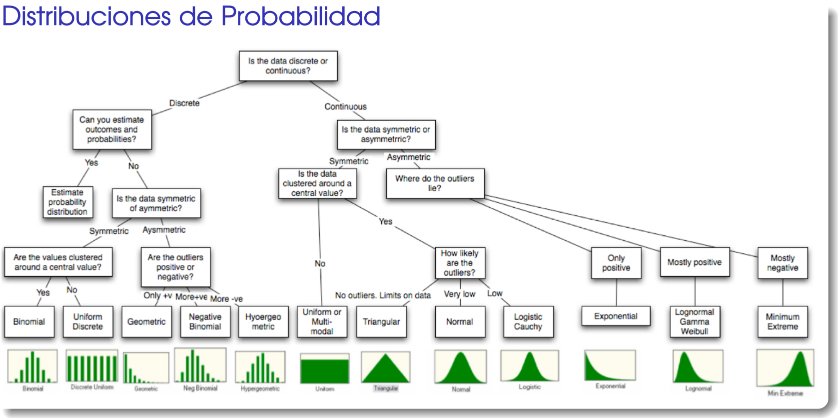
```
plot(random.number)
```



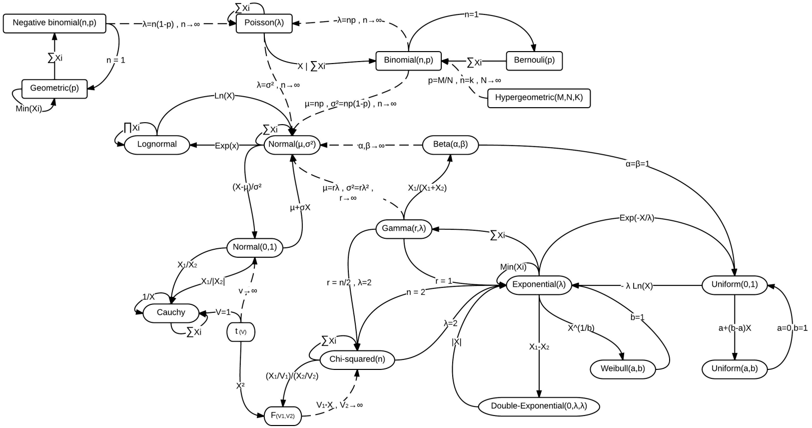
Distribuciones de Probabilidad

Una operación similar (usando otra fórmula y con un ciclo mucho más largo) es la que usa internamente R para producir números pseudoaleatorios de forma automática con la función `runif()` del grupo de funciones asociadas con la distribución uniforme, dentro del paquete `stats`. En este caso la semilla se selecciona internamente.

Distribuciones de Probabilidad

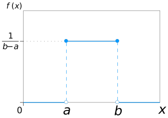


Relaciones entre las Distribuciones de Probabilidad



UNIVERSIDAD PERUANA CAYETANO HEREDIA
Escuela de Posgrado

Distribución uniforme

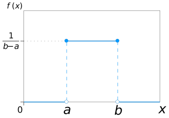


Descripción

Las siguientes funciones proporcionan información sobre la distribución uniforme en el intervalo comprendido entre `min` y `max`:

dunif	proporciona la función de densidad
punif	proporciona la función de distribución
qunif	proporciona la función de cuantiles
runif	genera valores aleatorios.

Distribución uniforme



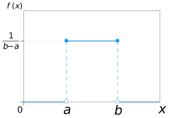
uso

```r
dunif(x, min=0, max=1, log = FALSE)
punif(q, min=0, max=1, lower.tail = TRUE, log.p = FALSE)
qunif(p, min=0, max=1, lower.tail = TRUE, log.p = FALSE)
runif(n, min=0, max=1)
```

- `x`, `q`: vector de cuantiles.
- `p`: vector de probabilidades.
- `n`: número de observaciones. Si no se especifica se toma igual a 1

UNIVERSIDAD PERUANA CAYETANO HEREDIA
ESCUELA DE POSGRADO

Distribución uniforme

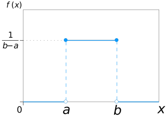


USO

```
dunif(x, min=0, max=1, log = FALSE)
punif(q, min=0, max=1, lower.tail = TRUE, log.p = FALSE)
qunif(p, min=0, max=1, lower.tail = TRUE, log.p = FALSE)
runif(n, min=0, max=1)
```

- `min,max`: extremos inferior y superior del intervalo que determina la distribución. Deben ser finitos. Si no se especifican se toman los valores por defecto 0 y 1. Para el caso min = max = u, el caso degenerado X = u se considera, aunque como no tiene función de densidad, la función `dunif` devuelve `NaN` (condición de error).

Distribución uniforme

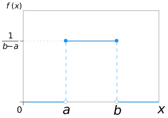


uso

```
dunif(x, min=0, max=1, log = FALSE)
punif(q, min=0, max=1, lower.tail = TRUE, log.p = FALSE)
qunif(p, min=0, max=1, lower.tail = TRUE, log.p = FALSE)
runif(n, min=0, max=1)
```

- `log`, `log.p`: son valores lógicos; si son TRUE, las probabilidades p se dan como probabilities log(p).
- `lower.tail`: es un valor lógico; si es TRUE (por defecto), las probabilidades son P(X $\leq$ x), en otro caso P(X $>$ x).

Distribución uniforme



USO

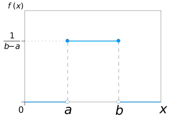
La forma de uso más habitual para generar números pseudoaleatorios de una distribución `U(a,b)` es

```
runif(n, min=a, max=b)
```

y para una `U(0,1)`

```
runif(n)
```

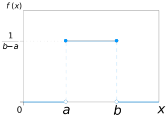
Distribución uniforme



Ejemplo : Generar 10 números aleatorios en el intervalo (0,1) y 15 en el intervalo (-1,2)

```
runif(10)
runif(15,min=-1,max=2)
```

Distribución uniforme



Obs. :

Si se quiere ejecutar la función anterior, pero partiendo de una semilla concreta (para garantizar el mismo resultado en cualquier ejecución), se usará la función `set.seed()`:

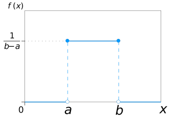
```
runif(5)
runif(5)
set.seed(32789)
runif(5)
set.seed(32789)
runif(5)
```

Ejercicio 1

Genera 1000 valores pseudoaleatorios usando la función runif() (con set.seed(19908)) y asígnalos a un vector llamado U.

a. Calcular la media, varianza y desviación típica de los valores de U.
b. Compara los resultados con los verdaderos valores de la media, varianza y desviación típica de una U(0,1).
c. Calcula la proporción de valores de U que son menores que 0.6 y compárala con la probabilidad de que una variable U(0,1) sea menor que 0.6.
d. Estimar el valor esperado de 1/(U+1)
e. Construir un histograma de los valores de u y de 1/(U+1)

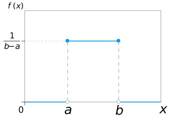
Distribución uniforme



Sol. : Ejercicio 1

```r
set.seed(19908)
U <-runif(1000)
data<-c(mean(U),var(U),sqrt(var(U)))
# 0.5 : media te rica
# 1/12 : varianza te rica
# sqrt(1/12) : desviaci n t pica te rica
#media, varianza y desviaci n t pica de los datos generados :
data
```

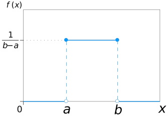
Distribución uniforme



Sol. : Ejercicio 1

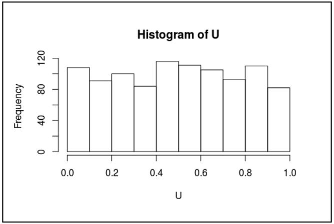
```
# La proporci n de valores menores que 0.6 se calcula como
sum(U<0.6)/1000

# o equivalentemente como
length(U[U<0.6])/length(U)

# La probabilidad te rica se calcula como
punif(0.6)

# Estimaci n del valor esperado de 1/(U+1)
mean(1/(U+1))
```

Distribución uniforme

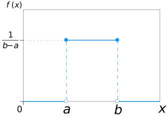


Sol. : Ejercicio 1

```r
hist(U)
```

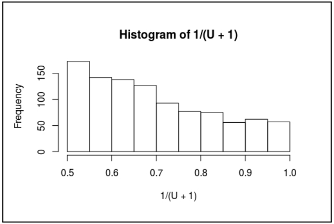


Distribución uniforme



Sol. : Ejercicio 1

```r
hist(1/(U+1))
```



Ejercicio 2

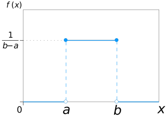
Simula 10000 observaciones independientes de una variable aleatoria distribuida uniformemente en el intervalo $(3.7, 5.8)$.

a. Calcular la media, varianza y desviación típica de los valores simulados (Estima la media, varianza y desviación típica de tal variable aleatoria uniforme) y compararlos con los verdaderos valores de la distribución.

b. Estima la probabilidad de que tal variable aleatoria sea mayor o igual que 4 (Calcula la proporción de valores que son mayores o iguales a 4) y compárala con el verdadero valor.

UNIVERSIDAD PERUANA
CAYETANO HEREDIA
Escuela de Posgrado

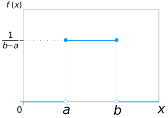
Distribución uniforme



Sol. : Ejercicio 2a

```
r<-runif(10000,3.7,5.8)
mean(r)
var(r)
sd(r)
# (3.7+5.8)/2  : media te rica
# (5.8-3.7)^2/12 : varianza te rica
# sqrt((5.8-3.7)^2/12) : desviaci n t pica te rica
```

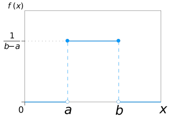
Distribución uniforme



Sol. : Ejercicio 2b

```r
length(r(r>4))/length(r)
punif(4, min=3.7, max=5.8, lower.tail = FALSE)
```

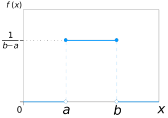
Distribución uniforme



Ejercicio 3

Simula 10000 valores de una variable aleatoria U_1 con distribución U(0,1) y otro conjunto de valores de una variable aleatoria U_2 con distribución U(0,1). Asignar esos valores a vectores `U1` y `U2`, respectivamente. Dado que los valores en `U1` y `U2` son aproximadamente independientes, podemos considerar a U1 y U2 variables aleatorias independientes U(0,1).

a. Estimar `E[U1+U2]`, compararla con el verdadero valor y compararla con una estimación de `E[U1]+E[U2]`,
b. Estimar `Var[U1+U2]` y `Var[U1]+Var[U2]`. ¿Son iguales? ¿Serían los verdaderos valores iguales?
c. Estimar `P(U1+U2 <= 1.5)`.
d. Estimar p($\sqrt{\texttt{U1}} + \sqrt{\texttt{U2}}) \leq 1.5$

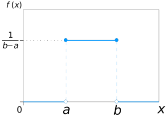


Distribución uniforme

Sol. : Ejercicio 3

```r
U1<-runif(10000,0,1)
U2<-runif(10000,0,1)
U=U1+U2   # definimos la v.a. U
# COMPARAR :
mean(U)
mean(U1)+mean(U2)
###############################
# COMPARAR :
var(U)
var(U1)+var(U2)
###############################
length(U[U<=1.5])/length(U)
V<-sqrt(U1)+sqrt(U2) # definimos la v.a. V
length(V[V<=1.5])/length(V)
```

Distribución uniforme



Ejercicio 4

Supongamos que `U1`, `U2` y `U3` son variables aleatorias independientes con distribución `U(0,1)`. Usa simulación para estimar las siguientes cantidades:

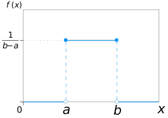
a. `E[U1+U2+U3]`

b. `Var[U1+U2+U3]` y `Var[U1]+Var[U2]+Var[U3]`

c. $\mathtt{E}[\sqrt{\mathtt{U1} + \mathtt{U2} + \mathtt{U3}}]$

d. $\mathtt{P}(\sqrt{\mathtt{U1}} + \sqrt{\mathtt{U2}} + \sqrt{\mathtt{U3}} \geq 0.8)$

UNIVERSIDAD PERUANA CAYETANO HEREDIA
Escuela de Posgrado



Distribución uniforme

Sol. : Ejercicio 4

```r
U1<-runif(10000)
U2<-runif(10000)
U3<-runif(10000)
U<-U1+U2+U3
# (a)
mean(U)
# (b)
var(U)
var(U1)+var(U2)+var(U3)
# (c)
mean(sqrt(U))
# (d)
V<-sqrt(U1)+sqrt(U2)+sqrt(U3)
length(V[V>=.8])/length(V)
```

FUNCIÓN SAMPLE

sample()

La función `sample()` permite tomar una muestra aleatoria simple a partir de un vector de valores con o sin reemplazamiento. Se usa como

```
sample(x, size, replace=FALSE, prob=NULL)
```

donde `x` es un vector de donde se quieren elegir los elementos o un entero positivo `n` (en este caso se interpreta como el vector generado por `1:n`), `size` es un entero positivo que indica el número de elementos que se quieren elegir, `replace=FALSE` indica que el muestreo se hace sin reemplazamiento, mientras que `replace=TRUE` indica con reemplazamiento. Por último en `prob` se puede incluir un vector de probabilidades en el que cada componente será la probabilidad con la que se elegirá la correspondiente componente del vector que va a ser muestreado .

FUNCIÓN SAMPLE

sample() : Ejemplo 1

```
sample(c(3,5,7), size=2, replace=FALSE)
```

conduce a un vector de dos valores tomados (sin reemplazamiento) del conjunto `{3,5,7}`.

FUNCIÓN SAMPLE

sample() : Ejemplo 1

Usar la función sample() para generar 50 números pseudoaleatorios del 1 al 100,

a. muestreados sin reemplazamiento.

b. muestreados con reemplazamiento.

```r
# a)
sample(1:100, size=50, replace=FALSE)
### Se podr a haber escrito simplemente
### sample(100,50,FALSE)
# b)
sample(1:100, size=50, replace=TRUE)
```

FUNCIÓN SAMPLE

sample() : Ejemplo 2

Simula el lanzamiento de un dado

```
sample(1:6,1)
sample(1:6,1)
sample(1:6,1)
sample(1:6,1)
```

UNIVERSIDAD PERUANA CAYETANO HEREDIA
ESCUELA DE POSGRADO

FUNCIÓN SAMPLE

sample() : Ejemplo 2

Simula el lanzamiento de cuatro dados o de un mismo dado cuatro veces

```
sample(1:6 ,4 ,  replace=T)
```

UNIVERSIDAD PERUANA CAYETANO HEREDIA
Escuela de Posgrado

FUNCIÓN SAMPLE

sample() : Ejemplo 2

Simula la distribución de la suma de los números que salen al lanzar cuatro dados .
Para ello usaremos la función sapply de la siguiente forma

```
t<-sapply(1:10000, function(x){sum(sample(1:6,4,rep=T))})
```

la cual aplica a un vector de tamaño 10000 una función sin nombre generando a su vez un vector de tamaño 10000. La función considerada obtiene muestras de tamaño y, a continuación, suma los elementos de la muestra. Se podría haber hecho con un `for` pero este procedimiento es más rápido.

FUNCIÓN SAMPLE

sample() : Ejemplo 2

Para garantizar que los resultados son los mismos usemos una semilla común,

```
set.seed(111)
t<-sapply(1:10000, function(x){sum(sample(1:6,4,rep=T))})
```

A continuación, tabulamos los resultados

```
table(t)
```

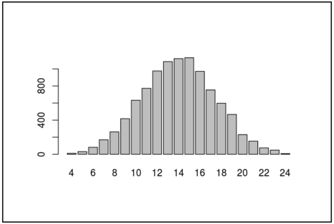
UNIVERSIDAD PERUANA CAYETANO HEREDIA
Escuela de Posgrado

FUNCIÓN SAMPLE

sample() : Ejemplo 2

y podemos representar los resultados con un diagrama de barras

```
barplot(table(t))
```



FUNCIÓN SAMPLE

sample() : Ejemplo 3

Supongamos una urna con 3 bolas blancas y 7 negras, simular la extracción de una bola (asignar, por ejemplo, el 1 a bola blanca y 0 a negra)

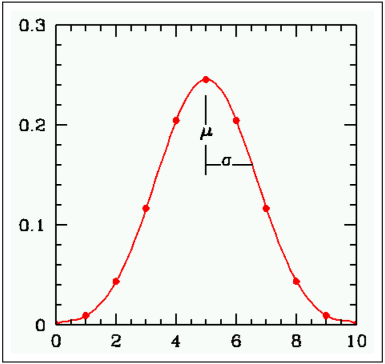
```
sample(c(1,0), 1, prob=c(0.3,0.7))
sample(c(1,0), 1, prob=c(0.3,0.7))
sample(c(1,0), 1, prob=c(0.3,0.7))
sample(c(1,0), 1, prob=c(0.3,0.7))
sample(c(1,0), 1, prob=c(0.3,0.7))
sample(c(1,0), 1, prob=c(0.3,0.7))
```

Si queremos simular 8 extracciones con reemplazamiento

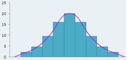
```
sample(c(1,0), 8, rep=T, prob=c(0.3,0.7))
```

FUNCIÓN SAMPLE

Si sólo nos interesara el número de bolas blancas que salen, se puede hacer la suma, pero esto lo haremos mejor usando la distribución binomial.



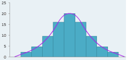
DISTRIBUCIÓN BINOMIAL



Descripción

Las siguientes funciones proporcionan información sobre la `distribucion binomial` de parámetros `size` (número de veces que se repite el experimento de Bernoulli) y `p` (probabilidad de éxito):

dbinom	proporciona la función masa de probabilidad.
pbinom	proporciona la función de distribución.
qbinom	proporciona la función de cuantiles.
rbinom	genera valores aleatorios.

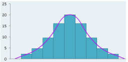
DISTRIBUCIÓN BINOMIAL 

Uso

```
dbinom(x, size, prob, log = FALSE)
pbinom(q, size, prob, lower.tail = TRUE, log.p = FALSE)
qbinom(p, size, prob, lower.tail = TRUE, log.p = FALSE)
rbinom(n, size, prob)
```

- `x, q`: vector de cuantiles.
- `p`: vector de probabilidades.
- `n`: número de observaciones. Si no se especifica se toma igual a 1
- `log, log.p`: son valores lógicos; si son TRUE, las probabilidades p se dan como probabilities log(p).
- `lower.tail`: es un valor lógico; si es TRUE (por defecto), las probabilidades son $P(X \leq x)$, en otro caso $P(X > x)$.

ANA HEREDIA

DISTRIBUCIÓN BINOMIAL 

Ejemplo 1 :

Calcular la probabilidad de obtener cuatro caras al lanzar seis veces una moneda perfecta.

En este caso sería P(X=4), con X $\rightarrow$B(6,0.5)

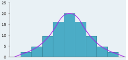
```
dbinom(x=4, size=6, prob=0.5)
```

Ejemplo 2 :

Calcular la probabilidad de obtener como mucho cuatro caras al lanzar seis veces una moneda perfecta

En este caso sería P(X$\leq$4), con X $\rightarrow$ B(6,0.5)

```
pbinom(q=4, size=6, prob=0.5)
```

DISTRIBUCIÓN BINOMIAL 

Ejemplo 3 :

Calcular el valor x tal que P(X $\leq$ x)=0.89

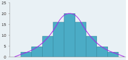
```
qbinom(0.89 ,6 ,0.5)
```

Ejemplo 4 :

Generar 10 valores pseudoaleatorios de una B(6,0.5)

```
rbinom(10 ,6 ,0.5)
```

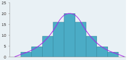
UNIVERSIDAD PERUANA
CAYETANO HEREDIA
Escuela de Posgrado



Ejemplo 5 :

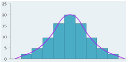
Supongamos que el 10% de los tubos producidos por una máquina son defectuosos y supongamos que produce 15 tubos cada hora. Cada tubo es independiente de los otros. Se juzga que el proceso está fuera de control cuando se producen más de 4 tubos defectuosos en una hora concreta. Simular el número de tubos defectuosos producidos por la máquina en cada hora a lo largo de un periodo de 24 horas y determinar si el proceso está fuera de control en algún momento.

```
TubosDefectuosos<-rbinom(24,15,0.1)
TubosDefectuosos
help(any)
any(TubosDefectuosos>4)
sum(TubosDefectuosos>4)
```

DISTRIBUCIÓN BINOMIAL 

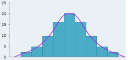
Ejercicio 1 :

Supongamos que en un proceso de manufactura la proporción de defectuosos es 0.15. Simular el número de defectuosos por hora en un periodo de 24 horas si se supone que se fabrican 25 unidades cada hora. Chequear si el número de defectuosos excede en alguna ocasión a 5. Repetir el procedimiento con p=0.2 y p=0.25.



Ejercicio 2 :

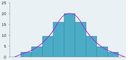
Simular 10000 números pseudoaleatorios de una variable aleatoria X con distribución B(20,0.3). Usar dichos valores para estimar $P(X \leq 5)$, $P(X=5)$, $E(X)$, $Var(X)$, el percentil 95, 99 de X.



Ejercicio 3 :

Usar simulación para estimar la media y la varianza de una variable aleatoria B(18,0.76) y comparar dichos valores con los teóricos.

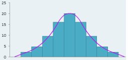
DISTRIBUCIÓN BINOMIAL



Método de inversión de la función de distribución Binomial

Considerar la siguiente función diseñada para simular valores pseudoaleatorios de una distribución binomial usando el llamado método de inversión:

```
ranbin<-function(n,size,prob){
cumbinom<-pbinom(0:(size-1),size,prob)
singlenumber<-function(){
x<-runif(1)
N<-sum(x>cumbinom)
N
}
replicate(n,singlenumber())
}
```

DISTRIBUCIÓN BINOMIAL 

Ejercicio 4 :

Usar ranbin() `[Slide anterior]` para similar vectores de longitud 1000, 10000 y 100000 de una distribución B(10,0.5). Usar la función system.time() para comparar los tiempos de ejecución para esas simulaciones con los tiempos de ejecución correspondientes cuando se usa rbinom().