Universidad Nacional de Colombia

Especialización en Analítica

Módulo Simulación y optimización

Tema:

GENERACIÓN DE VARIABLES ALEATORIAS EN R

Elaborado a partir de: Banks et al. Discrete Event System Simulation. 4 ed. Pearson, 2005.

R es un lenguaje de programación que tiene funciones estadísticas incorporadas en paquetes de distribución gratuita.

Con R se pueden resolver problemas estadísticos usando menos comandos que en otros lenguajes

1. Generación de variables distribuidas uniformemente

La mayoría de los lenguajes de programación tienen rutinas que generan muestras de la distribución uniforme[0,1].

En R la función

```
U=runif(n, min=0, max=1)
```

regresa un valor seudoaleatorio muestreado de la función uniforme en el intervalo abierto (0,1).

2. Distribución Bernoulli

Una prueba con dos resultados (éxito o fracaso) se repite n veces.

Xj = 1 si el j-ésimo experimento es un éxito y Xj=0 si el j-ésimo experimento es un fracaso. Si las n réplicas del experimento son independientes y la probabilidad de éxito/fracaso permanece constante,

p(x1, x2, ..., xn)=p1(x1)p2(x2)....pn(xn)

y pj(xj)=p(xj)= $\begin{cases} p, & xj = 1,\ j = 1,2,..n \\ 1 - p = q, & xj = 0,\ j = 1,2,..n \\ 0, & \text{de otra forma} \end{cases}$

Ejemplo 2.1 Simular el lanzamiento de una moneda con probabilidad p de que salga cara.

Si U~ U(0,1), la variable aleatoria X~ Bernoulli como

X={1 para U<p || 0 para U>=p}

P(cara)=P(X=1)=P(U<p)=p

```
# Esta función hace un experimento Bernoulli con n pruebas

Bernoulli<- function(probs,n){
  u=runif(n, min=0, max=1);
  return(u<probs);
}
```

Ejemplo 2.2

```
# Este código hace un experimento Bernoulli con n=1000 pruebas y grafica los resultados.

n =1000;
a = numeric(n + 1);
u=runif(n, min=0, max=1);
toss= u<0.5;
avg = numeric(n);
for(i in 2 : n + 1)
{
a[i] = a[i-1] + toss[i-1];
avg[i-1] = a[i]/(i-1);
}
```

```
plot(1:n,avg[1:n],type = "l", lwd = 5, col = "blue",ylab = "ProportionofHeads", xlab = "CoinTossNumber",cex.main =1.25,cex.lab = 1.5,cex.axis = 1.75)
```

3. Distribución Binomial

La variable aleatoria X que denota el número de éxitos en n pruebas independientes Bernoulli tiene una distribución binomial dada por p(x) donde

$$p(x) = \begin{cases} \binom{n}{x} p^x q^{n-x} & x = 0,1,2,\dots,n \\ 0, & de\ otra\ forma \end{cases}$$

La probabilidad de tener x éxitos seguidos de n-x fracasos es p^xq^(n-x) y hay n!/(x!(n-x)!)formas de tener el número requerido de éxitos y fracasos.

Ejemplo 3.1 Generación de variables aleatorias binomiales

Se puede usar que si X1, X2, ...Xn son variables aleatorias independientes con una distribución Bernoulli (p), la variable aleatoria X definida por X=X1+X2+,...,+Xn tiene una distribución binomial (n, p)

Generar una variable aleatoria X~binomial(50, 0.02)

```
Binomial<- function(probs,reps){

probs=0.02;

reps= 50;

u=runif(reps, min=0, max=1);

x=sum(u<probs);

return(x);

}
```

4. Generación de variables aleatorias discretas (General)

Queremos simular la variable aleatoria discreta X con rango

Rx={x1, x2, ..., xn} y P(X=xj)=pj tal que $\sum_j p_j = 1$.

Para lograr esto el procedimiento es:

i. Generar un número aleatorio U~Uniforme(0,1).
ii. Dividir [0,1] en subintervalos tales que el j-ésimo subintervalo tenga longitud pj
iii. Asumir que

$$X = \begin{cases} x_0 \quad si\ (U < p_0) \\ x_1 \quad si\ (p_0 \leq U < p_0 + p_1) \\ \vdots \\ x_j \quad si\ (\sum_{k=0}^{j-1} p_k \leq U < \sum_{k=0}^{j} p_k) \\ \vdots \end{cases}$$

Es decir, $X=x_j$ si $F(x_{j-1}) \leq U < F(x_j)$. (F(x) es la FDP Acumulada)

Ejemplo 4 Función discreta.

El tiempo transcurrido entre el pedido de una moto y su entrega al distribuidor se distribuye de la siguiente manera

Tiempo (días)	p(x) probabilidad	probabilidad acumulada F(x)	intervalo
1	0.6	0.6	[0,0.6)
2	0.3	0.9	[0.6,0.9)
3	0.1	1	[0-9,1)

4.1 Generar una variable aleatoria con distribución discreta

```
# Código en R

# P es un vector que contiene las probabilidades

# X es un vector que contiene los valores que toma X

P=c(0.6, 0.9, 1);

X=c(1,2,3);

Counter=1;

# generar número aleatorio

r=runif(1,min=0,max=1);

# counter es una variable auxiliar para determinar si r está en  el
intervalo i

while (r>P[counter])
```

```
    counter=counter+1;
end
X[counter]
```

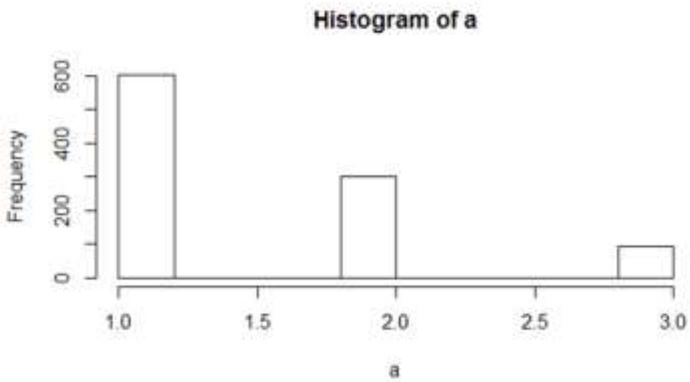
Ejemplo 4.2. Función para muestrear de la fdp discreta.

```
# Función para muestrear de la distribución discreta
# n es el tamaño de muestra
# p es un vector 1..3 con las probabilidades
# val es un vector 1..3 con los valores que toma la variable aleatoria

rx<- function(n,p,val)
{
  # generar n números aleatorios
  r=runif(n,min=0,max=1);
  # almacenar las variables generadas en X
  # la función rep()replica los valores en x
  # rep(0,n) crea un vector de longitud n y lo llena de ceros
  X <- rep(0,n);
  # ciclo para revisar la secuencia r
  for(i in 1:n)
  {
     # counter es una variable auxiliar para determinar si r está en
     el intervalo i
    counter=1;
    while (r[i]>p[counter])
      counter=counter+1;
    end
    X[i]=val[counter];
  }
  return(X)
}
```

Generar 1000 números y verificar

```
> a<-rx(1000,c(0.6,0.9,1),c(1,2,3))
> hist(a)
> mean(a==1)
[1] 0.604
> mean(a==2)
[1] 0.301
> mean(a==3)
[1] 0.095
>
```



5. Generación de números aleatorios que siguen una distribución exponencial.

La función de densidad de probabilidad exponencial es:

$$f(x) = \begin{cases} \lambda e^{-\lambda x}, & x \geq 0 \\ 0, & 0 < x \end{cases}$$

Y la distribución acumulada es:

$$F(x) = \int_{-\infty}^{x} \lambda e^{-\lambda t} dt = \begin{cases} 1 - e^{-\lambda x}, & x \geq 0 \\ 0, & 0 < x \end{cases}$$

El parámetro λ se puede interpretar como el promedio de ocurrencias por unidad de tiempo. Por ejemplo, para tiempos entre llegadas X1, X2, ..Xn que se distribuyen exponencialmente con tasa λ, λ puede interpretarse como el número promedio de llegadas por unidad de tiempo (tasa de llegadas).

Para generar variables aleatorias que se distribuyen exponencialmente con parámetro λ, se puede invertir la función de probabilidad acumulada F(x):

Sea F(x)=R en el rango de x

$$F(X) = R = 1 - e^{-\lambda X}$$

Resolviendo la ecuación F(X)=R en términos de X se tiene:

$$X = -\frac{1}{\lambda}\ln(1-R)$$

Genere números aleatorios R1, R2, ..., Rn y calcule las variables alea torias deseadas como:

$$X_i = -\frac{1}{\lambda}\ln(1-R_i)$$

Como (1-Ri)~U(0,1), la expresión anterior se suele simplificar como

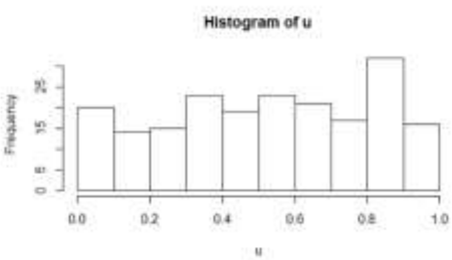
$$X_i = -\frac{1}{\lambda}\ln R_i$$

Ejemplo 5: generación de n números que sigan la fdp exponencial con λ= 1

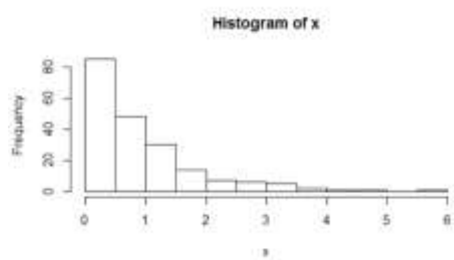
```
# Función

gen_expo <-function(n, lambda)
{
u=runif(n, min=0, max=1);
x=-log(u);
return(x)
}
```

para n=200, la distribución de u es:



Y la distribución de x es:



El método descrito puede usarse para generar variables aleatorias cuya distribución acumulada sea lo suficientemente sencilla como para resol ver F(X)=R en términos de X, encontrando $X=F^{-1}(R)$.

6. Generar números aleatorios con distribución normal

Para generar variables aleatorias distribuidas normalmente se usa el m étodo de Box-Muller

Box-Muller (Ann. Math. Statist. Volume 29, Number 2 (1958), 610-611.): Se puede gener ar un par de variables normales independientes (Z1, Z2), transformando un par de variables aleatorias uniformes (U1, U2) e independientes:

$$Z_1 = (-2lnR_1)^{\frac{1}{2}}\cos(2\pi R_2)$$

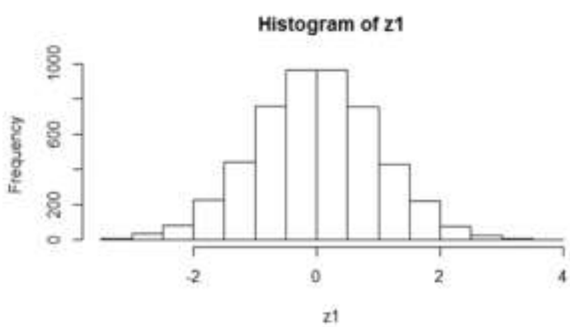
$$,Z_2 = (-2lnR_1)^{1/2}\sin(2\pi R_2)$$

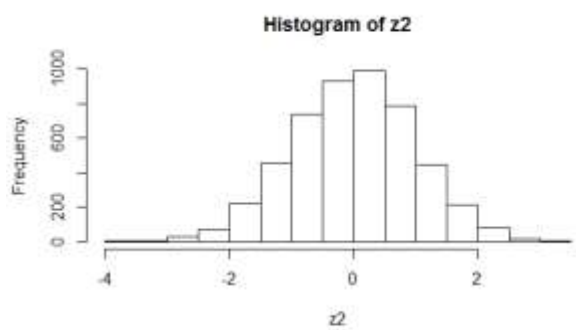
Para obtener variables Xi con media μ y varianza σ^2, se aplica la trans formación:

$$X_i = \mu + \sigma Z_i$$

Ejemplo 6: generar 5000 pares de variables aleatorias normales y grafi car su histograma

```
n=5000;
u1=runif(n,min=0,max=1);
u2=runif(n,min=0,max=1);
z1=sqrt(-2*log(u1))*cos(2*pi*u2);
z2=sqrt(-2*log(u1))*sin(2*pi*u2);
hist(z1);
hist(z2);
```





7. Funciones en R

R tiene comandos para las funciones de probabilidad más comunes. Los c omandos tienen un prefijo que indica qué función calculan, así:

p: función de probabilidad acumulada ,
q: función de probabilidad acumulada inversa
d: función de densidad de probabilidad
r: una variable aleatoria con la distribución especificada.

$Binomial$	$pbinom$	$qbinom$	$dbinom$	$rbinom$
$Geometric$	$pgeom$	$qgeom$	$dgeom$	$rgeom$
$NegativeBinomial$	$pnbinom$	$qnbinom$	$dnbinom$	$rnbinom$
$Poisson$	$ppois$	$qpois$	$dpois$	$rpois$
$Beta$	$pbeta$	$qbeta$	$dbeta$	$rbeta$
$Beta$	$pbeta$	$qbeta$	$dbeta$	$rbeta$
$Exponential$	$pexp$	$qexp$	$dexp$	$rexp$
$Gamma$	$pgamma$	$qgamma$	$dgamma$	$rgamma$
$Studentt$	pt	qt	dt	rt
$Uniform$	$punif$	$qunif$	$dunif$	$runif$

8. Función SAMPLE

La función sample() toma una muestra de tamaño especificado de los elementos de x, con o sin reemplazo.

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

Argumentos:
x: entero positivo, o vector de uno o más elementos de los cuales escoger-
n: número positivo, el número de ítems de los cuales escoger.
size: entero no negativo con el número de ítems a escoger.
replace: true/false con o sin reemplazo.
prob: vector con pesos de probabilidad para obtener los elementos del vector de donde se saca la muestra.

Ejemplo 7.1 para hacer 100 pruebas Bernoulli (1: éxito, 0:fracaso)

```
# 100 Pruebas Bernoulli
sample(c(0,1), 100, replace = TRUE)
```

Ejemplo 7.2 simular el resultado de lanzar 3 dados (suma)

```
##Esta función usa sample para lanzar tres dados y sumar el resultado.
tres_dados<- function() {
dados<-sample(1:6, size=3, replace=TRUE)
return (sum(dados))
}

## REPLICATE reproduce el experimento
## replicate(n, expr). n es el número de veces
## expr es la función

## para 100 reproducciones se guarda la suma en
## el vector sims
sims<-replicate(n=100, tres_dados())
```

```
## la función TABLE calcula las frecuencias
## para calcular las frecuencias relativas
## usamos la función length() que calcula n=100
table(sims)/length(sims)

## graficando el histograma
plot(table(sims), xlab='suma', ylab= 'frecuencia', main='suma 3 dados,
100 repeticiones')

#y la frecuencia relativa
plot(table(sims)/length(sims), xlab='suma', ylab= 'frecuencia relativa
', main='suma 3 dados, 100 repeticiones')
```