# Pràctica 4. Simulació de variables aleatòries

## 1 Introducció

La simulació és una eina important en la modelització de fenòmens aleatoris quan el càlcul de probabilitats exacte és difícil de dur a terme. També és de gran utilitat per desenvolupar la intuïció estadística.

Bàsicament, *simular* vol dir usar l'ordinador per generar valors de variables aleatòries, de manera que puguem fer càlculs sense haver d'observar el fenomen aleatori real.

Hem vist que l'R ens permet generar amb una sola instrucció valors de variables aleatòries seguint certes lleis de probabilitat. Per exemple,

```
runif(min=0, max=1, n=100);
```

genera 100 valors d'una variable uniforme a l'interval $[0, 1]$. Podem representar els valors que han sortit mitjançant la funció gràfica `rug()`. Com que és una funció gràfica "de baix nivell", hem de crear primer un `plot()` per poder-la usar. Però el plot pot ser buit. Això s'aconsegueix amb l'opció `type="n"`, que vol dir "no dibuixis":

```
plot(0,0, xlim=c(0,1), ylim=c(0,1), type="n");
mostra=runif(min=0,max=1, n=100);
rug(mostra);
```

Els `0,0` hi són perquè alguna cosa s'hi ha de posar. Qualsevol cosa produirà el mateix resultat. La funció `rug()` permet visualitzar com es reparteixen els valors d'una mostra mitjançant els "pèls" d'una "manta".

### Problema 1

Dibuixeu en un mateix gràfic la densitat d'una normal estàndard, el resultat d'una mostra simulada de mida $n = 100$ amb `rnorm()` mitjançant un histograma, i els valors de la mostra mitjançant `rug()`. Fer el dibuix entre -4 i 4 segurament serà suficient.
Executeu el mateix codi sencer diverses vegades per visualitzar les diferents configuracions que poden sortir quan agafem diferents mostres a l'atzar d'aquesta població normal.

```
mostra <- rnorm( mean=0, sd=1, n=100);
hist( mostra, breaks=seq(-4,4,0.1), freq=FALSE);
lines(seq(-4,4,0.1),dnorm(seq(-4,4,0.1), mean = 0, sd = 1, log = FALSE));
```

Com ho fa l'R per simular mostres? Els ordinadors només poden executar algorismes preprogramats; i això implica que no poden produir res realment aleatori. En efecte, els ordinadors no poden inventar mostres aleatòries. Es fa trampa, però és una trampa que funciona. És possible generar seqüències de números entre 0 i 1 mitjançant un algorisme que siguin indistingibles, a tots els efectes pràctics, d'una seqüència de valors seguint una llei uniforme en $[0, 1]$.

Hi ha diferents mètodes; l'R usa per omissió un mètode anomenat *Mersene-Twister* (feu `?RNG` a la consola per a més informació, o `https://en.wikipedia.org/wiki/Mersenne_Twister`). Tots els mètodes entren en un cicle tard o d'hora (comencen a repetir els mateixos números). Els generadors són bons si el seu cicle és gran. En el cas del *Mersene-Twister*, el cicle és de $2^{19937} - 1$ valors, que és una enormitat.

Com haureu observat, les mateixes instruccions donen seqüències diferents quan s'executen diverses vegades o en ordinadors diferents. Això és degut a que tots els mètodes usen un paràmetre, anomenat *llavor*, per inicialitzar la seqüència. Aquest paràmetre el crea l'R combinant el que marca el rellotge intern de l'ordinador amb el *número de procés* amb què el sistema operatiu identifica al propi R com a programa en execució. Per això cada cop surten valors aleatoris diferents.

De vegades, però, convé que les seqüències aleatòries siguin *reproduïbles*. Per exemple, si estem estudiant un sistema que té un paràmetre controlable i volem trobar el valor òptim d'aquest paràmetre, ens convé provar diferents valors exactament en les mateixes condicions. Per tant, volem poder generar els mateixos valors aleatoris.

Això es fa establint explícitament una llavor abans de cridar la funció pertinent. La llavor és qualsevol número enter entre $-(2^{31} - 1)$ i $(2^{31} - 1)$.

```
set.seed( 27092015);
print( rexp( rate=3, n=5));
set.seed( 27092015);
print( rexp( rate=3, n=7));
```

Observeu que les dues seqüències comencen igual. En canvi, si fem

```
set.seed( 27092015);
print( rexp( rate=3, n=5));
print( rexp( rate=3, n=7));
```

el segon cop l'R ha buscat una llavor pel seu compte.

A la natura *hi ha* fenòmens realment aleatoris, i és possible usar-los per a generar nombres aleatoris de veritat. Existeixen, des del 2001, petits aparells connectables per USB a l'ordinador, capaços de produir de l'ordre de 4 milions de bits aleatoris per segon. Es basen en l'emissió de fotons sobre un mirall semitransparent inclinat 45° respecte el raig de llum. Amb probabilitat 0.5 el fotó travessa el mirall; amb probabilitat 0.5 es reflexa perpendicularment, tal com prediu la mecànica quàntica. A partir de la seqüència de zeros i uns equiprobables que es produeix, és fàcil obtenir números amb llei uniforme en $[0, 1]$ sense més que tallar la seqüència a trossos, segons els decimals que es vulguin, i passar de base 2 a base 10 cada tros.

Hi ha també llocs web que ofereixen "on-line" seqüències de bits que hom pot descarregar. Per exemple, en `http://qrng.anu.edu.au`, que utilitza un mètode quàntic diferent (les fluctuacions quàntiques del buit) s'ha obtingut la seqüència següent:

```
11111100101100011001010011101100100001101100000011010010010011010100011110011101
01110001011000101011000011001110110111111111111011100110001000100111100000001000
00101000010100011110010010111000001110111011100101011101010110000100011111100010
00011100001101111001010011001111100111100111000110011001011010011101011000011111
01110011110010011111011000101011101001001110110111111011001001001101010101011100
00000011000100101110111001010011001110011111001110110111111100000110110110110101
11111000110100100100 11
```

## Curiositat

Connecteu-vos a la web mencionada i veieu en pantalla com es van generant els números. Aneu a `Live Numbers -> Live Streams -> Binary` per veure bits. Però hi ha altres possibilitats, convertint els números en soroll, colors, combinacions de loteria, etc.

La base per generar valors de variables aleatòries amb qualsevol distribució és tenir un bon generador de variables uniformes en $[0, 1]$. L'R el té; encara que no usem un generador quàntic podem perfectament suposar que partim d'una mostra potencialment infinita de valors d'uniformes en $[0, 1]$.

# 2 Simulació d'una variable aleatòria pel mètode de la transformació inversa

Abans d'explicar la base del mètode, definim el concepte de funció de distribució inversa generalitzada. Donada una funció de distribució $F$, definim la **funció de distribució inversa generalitzada** com

$$F^{-1}(u) := \min\{x \,:\, F(x) \geq u\}.$$

El mètode de la transformada inversa es basa en la següent propietat: donada una funció de distribució $F$, la variable aleatòria $F^{-1}(U)$ on $U \sim \text{Unif}[0, 1]$ té precisament $F$ com a funció de distribució. Per tant, si coneixem la funció de distribució $F_X$ d'una variable aleatòria $X$, aleshores podem simular realitzacions de $X$ utilitzant el fet que $X \sim F_X^{-1}(U)$. L'algoritme general és:

**Algoritme 1.**

1. Generar un valor $u$ de $U \sim \text{Unif}[0, 1]$.
2. $x = F_X^{-1}(u)$

El principal avantatge d'aquest mètode és que és bijectiu, és a dir, per a cada $U$ generada obtenim una i només una $X$. Això fa que sigui molt ràpid, tant ràpid com generar una realització de $U$ i calcular-ne la seva antiimatge "generalitzada"per la funció de distribució. Per contra, sovint no serà possible expressar $F_X^{-1}$ en termes de funcions elementals. Això és el que passa, per exemple, amb la $\text{N}(\mu, \sigma^2)$. En aquests casos es pot calcular $F_X^{-1}$ numèricament a cada pas (cosa que pot reduir molt la velocitat del mètode) o utilitzar una certa funció que aproximi bé $F_X^{-1}$. L'altre mètode que veurem en aquesta pràctica evita passar per $F_X^{-1}$, a costa d'augmentar el temps de còmput.

## 2.1 Variables aleatòries discretes

Generar variables amb una quantitat finita de valors és molt simple. L'Algoritme 2 equival a dividir l'interval $[0, 1]$ en subintervals que tinguin per longitud la probabilitat de cada valor. Per exemple, si volem mostrejar d'una variable aleatòria $X$ tal que $\mathbf{P}(X = 1) = 0.2$, $\mathbf{P}(X = 2) = 0.3$, $\mathbf{P}(X = 3) = 0.5$, faríem

```
u <- runif( min=0, max=1, n=100);
x <- cut( u, breaks=c( 0, 0.2, 0.5, 1), labels = c(1,2,3));
print( x);
```

Si no es vol que aparegui el `levels: 1,2,3` en el resultat, es pot fer

```
print( as.integer(x));
```

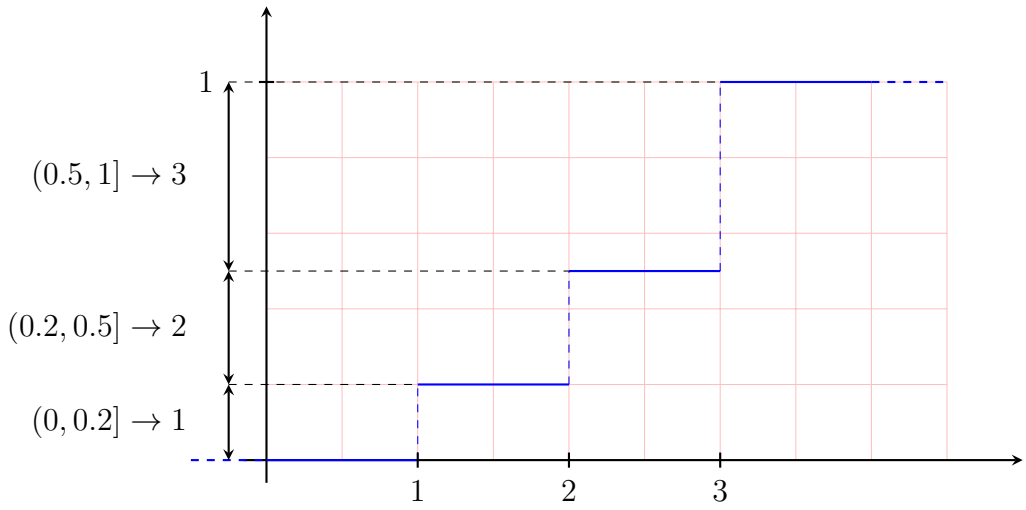
i per comprovar que estem fent el que toca observem que

```
x <- as.integer(x);
pobservades <- c(sum(x==1),sum(x==2),sum(x==3))/length(x);
print( pobservades);
```

Això és precisament el que fa la funció `sample()`, que vam utilitzar a la pràctica 2:

```
sample( c(1,2,3), size=100, replace=TRUE, prob=c( 0.2, 0.3, 0.5));
```

En el fons, el que estem fent és invertir la funció de distribució de la variable aleatòria:




Per tant, una tercera manera de fer el mateix és calculant la antiimatge "generalitzada"de la funció de distribució:

```
u <- runif( min=0, max=1, n=100);
p <- c(0.2, 0.3, 0.5);
F <- c(0,cumsum(p));
x <- 0;
for (i in 1:3) {x <- x+((F[i]< u)&(u <=F[i+1]))*i};
print(x);
pobservades <- c(sum(x==1),sum(x==2),sum(x==3))/length(x);
print( pobservades);
```

El gràfic anterior ajuda a entendre l'algoritme general següent per generar una variable aleatòria discreta $X$ amb llei $\mathbf{P}(X = x_i) = p_i$ i amb funció de distribució $F$.

**Algoritme 2.**

1. Generar un valor $u$ de $U \sim \text{Unif}[0, 1]$.
2. Si
$$F(x_{i-1}) = \sum_{k=1}^{i-1} p_k < u \leq \sum_{k=1}^{i} p_k = F(x_i)$$
fer $x = x_i$.

## Problema 2

Considereu $p \in (0, 1)$ i la variable aleatòria $X$ amb funció de probabilitat

$$\mathbf{P}(X = k) = \frac{p^k}{C} \qquad \text{on} \qquad C = \sum_{k=0}^{\infty} p^k = \frac{1}{1-p}.$$

Per $p = 0.3$ genereu una mostra de 100 valors.

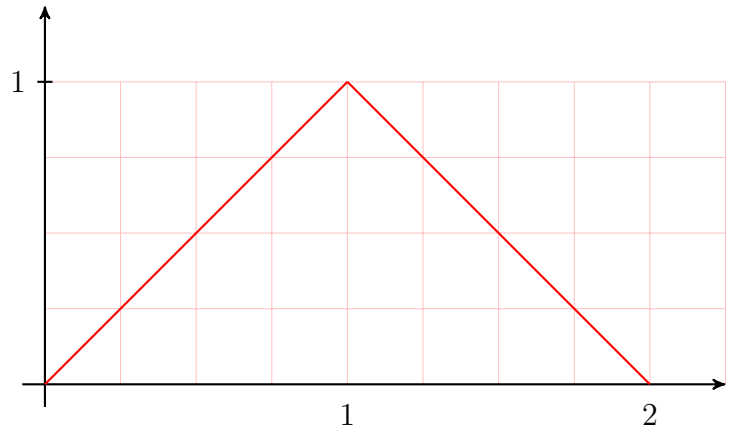
```
u <- runif( min=0, max=1, n=100)
x <- 0
p <- 0.3
F <- function(i){1-p^i}
assolits <- 0
i <- 0
while (assolits < 100) {
    x <- x+((F(i)< u)&(u <=F(i+1)))*i;
    assolits <- assolits + sum((F(i)< u)&(u <=F(i+1)))
    i<-i+1
}
print(x)
pobs <- p^c(0:max(x))*(1-p);
pemp <- rep(0,max(x));
for(i in 0:max(x)) {pemp[i+1] <- sum(x==i)/length(x)};
print(abs(pobs-pemp))
```

# 3 Simulació d'una variable aleatòria pel mètode d'acceptació-rebuig

Anem a veure un mètode molt general per generar mostres de lleis de probabilitat en aquesta situació. Com a exemple fàcil, considereu una densitat "triangular" (en aquest cas, sabríem calcular $F^{-1}$, però

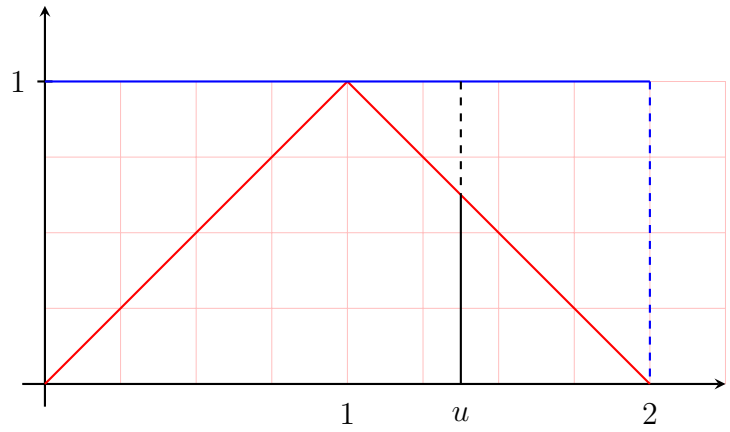
suposem que no).

$$f(x) = \begin{cases} x\,, & 0 \leq x \leq 1 \\ 2 - x\,, & 1 \leq x \leq 2 \end{cases}$$




Emboliquem la densitat amb un rectangle de base $[0, 2]$ i alçada $[0, 1]$. Aquest rectangle és la densitat d'una uniforme en $[0, 2]$ multiplicada per una constant $C$. Concretament, $C = 2$.

Ara generem un valor $u$ d'una uniforme en $[0, 2]$ i tracem un segment vertical sobre aquest valor:




Acceptarem $u$ com a valor de la densitat triangular $f$ amb una probabilitat que serà la proporció de segment que cau sota el triangle, respecte la longitud total del segment. En aquest cas, només ens cal generar una altre valor uniforme $v$ en $[0, 1]$ "sobre l'eix vertical", i veure si aquest valor és menor o no que $f(u)$.

Aquest procediment és un cas particular del mètode d'acceptació-rebuig que es dóna tot seguit. Sigui $X$ una variable aleatòria amb densitat $f_X(x)$ i $V$ una altra variable aleatòria amb densitat $f_V(v)$. Suposem que la variable $V$ és fàcil de simular (en l'exemple anterior era una uniforme!), i que existeix una constant $C \geq 1$ tal que

$$\frac{f_X(x)}{f_V(x)} \leq C \quad \forall x \in \mathbb{R}.$$

Aleshores, podem simular $X$ mitjançant el següent algoritme.

**Algoritme 3.**

1. Generar realitzacions $u$ i $v$ de variables $U \sim \text{Unif}[0,1]$ i $V$ independents.
2. Si $u > \frac{f_X(v)}{C f_V(v)}$ torna al pas 1. Si no, passa al pas 3.
3. Fer $x = v$

Per a provar que realment aquest algoritme funciona, s'ha de provar que $\mathbf{P}(V \leq v \mid B) = \mathbf{P}(X \leq v)$, on $B = \{CU \leq \frac{f_X(V)}{f_V(V)}\}$.

A continuació comprovarem que l'algoritme funciona generant una mostra de la densitat triangular i dibuixant-ne un histograma.

i) Definiu una funció de `R` que correspongui a la densitat triangular:

```
dens.triangular <- function(x){
  if( (0<=x) && (x<=1) ) return( x)
  if( (1<=x) && (x<=2) ) return( 2-x)
  if( (x<0) && (x>2) ) return( 0)
  }
```

ii) Definiu una funció de `R` que vagi generant valors $u$ fins que un d'ells s'accepti:

```
acceptacio <- function(){
  while ( TRUE){
    u <- runif( min=0, max=2, n=1)
    v <- runif( min=0, max=1, n=1)
    if( v < dens.triangular( u)) return( u)
    }
  }
```

iii) Genereu una mostra de 1000 valors:

```
mostra <- rep( 0, 1000) # de moment, posem tot zeros
for( i in 1:1000){
  mostra[i] <- acceptacio()
  }
```

iv) Finalment, dibuixeu l'histograma de la mostra obtinguda, superposant la densitat triangular:

```
hist( mostra, breaks=seq( from=0, to=2, by=0.1), freq=FALSE, ylim=c( 0, 1.5))
points <- seq( from=0, to=2, by=0.01)
densitat <- c()
for ( x in points){
  densitat <- c( densitat, dens.triangular( x))
  }
lines( points, densitat)
```

Adapteu el programa anterior per generar una mostra de mida qualsevol: Poseu una instrucció `mida <- 1000` al principi de tot, i useu la variable `mida` enlloc de `1000` a la resta del programa. Executeu-lo amb diferents valors de `mida` per veure si l'histograma es va adaptant més a la funció de densitat.

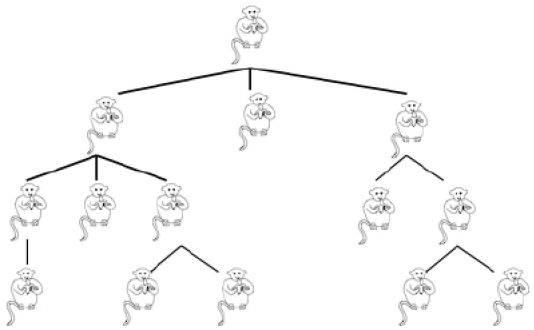
# 4 Generant variables aleatòries associades a sistemes complexos

L'interès més important de la simulació com a metodologia és la possibilitat de generar variables molt més complexes que les lleis estàndard, i sobretot que evolucionen en el temps (*processos estocàstics*).

**Simulació d'un procés de Galton-Watson.** El procés de Galton-Watson va sorgir com a model per a l'extinció de cognoms (de les famílies aristocràtiques a l'Anglaterra del segle XIX), però es pot utilitzar en qualsevol situació on es produeixi una ramificació (organismes vius, epidèmies, propagació de tumors, virus informàtics o cert tipus de cues).

L'ingredient bàsic és una llei de probabilitat $\{p_k\}$ en els enters no negatius $k = 0, 1, 2, \ldots$.

- La població comença amb un progenitor que constitueix la *generació* 0.
- Aquest progenitor inicial dóna origen a $k$ descendents amb probabilitat $p_k$.
- Aquests descendents formen la *generació* 1, que té un nombre (aleatori) $Z_1$ d'individus.
- Cada individu de la primera generació es reprodueix independentment i té un nombre aleatori de descendents, determinat per $\{p_k\}$, que constitueixen la *generació* 2.
- Aquest procés continua indefinidament, fins a l'extinció (si s'arriba a produir), que ocorre quan cap dels membres d'una generació té descendència.



$$
\begin{aligned}
Z_0 &= 1 && \text{número d'individus de la generació 0}\\
Z_1 &= Z_{1,1} && \text{número d'individus de la generació 1}\\
Z_2 &= Z_{2,1} + \cdots + Z_{2,Z_1} && \text{número d'individus de la generació 2}\\
&\vdots\\
Z_n &= Z_{n,1} + \cdots + Z_{n,Z_{n-1}} && \text{número d'individus de la generació } n
\end{aligned}
$$

on $Z_{i,j} \sim p_k$ és el número d' individus de la generació $i$ que són descendents de l'individu $j$ de la generació $i-1$. Observeu que $Z_n = 0 \Rightarrow Z_{n+1} = 0$, i que $Z_{n-1}$ és independent de $\{Z_{n,j}, j \geq 1\}$.

Suposem que la distribució de la descendència ve donada per

$$\mathbf{P}(Z_{i,j} = k) = (1-p)^k p \qquad k = 0, 1, \ldots$$

amb $p = 0.35$.

## Problema 3

L'objectiu és fer un programa que simuli aquest procés fins a la generació $n$ (amb $n = 1000$ fixat).

1. Feu una taula que mostri el nombre d'individus en les 10 primeres generacions d'aquest procés, per a cadascuna de les 10 primeres simulacions. Què crida l'atenció del comportament del procés? (Fila: generació, col.lumna: rèplica)
2. Estima la probabilitat que el procés s'extingeixi abans de la generació $n = 10$.
3. Calcula l'esperança i la variància del nombre d'individus de les generacions 5 i 10.
4. Fes un histograma de la distribució de $Z_n$ per a $n = 5$ i $n = 10$.
5. Fixa ara $p = 1/2$ i calcula la probabilitat que el procés s'extingeixi abans de la desena generació.
6. Fes un histograma de $Z_{10}$. Què s'observa?

**Nota 1:** La suma de variables geomètriques independents és una variable binomial negativa, la qual es genera amb al comanda `rnbinom`, amb els paràmetres adients.

**Nota 2:** `options(warn=-1)` evita que apareguin warnings a la consola. Hi haurà warnings quan s'intenti generar una binomial negativa amb $n = 0$, corresponent als descendents d'una rama en la que ja no hi queda ningú.