Técnicas de Simulación

01. Simulación estadística

Miguel A. Castellanos

Contents

En esta semana vamos a empezar a realizar pequeñas simulaciones, vamos a empezar por las simulaciones estadísticas (*Montecarlo*) porque debido a la temática del máster nos sentiremos más cómodos con ellas. Intentaremos comprobar algunos de los teoremas más conocidos de la estadística inferencial.

El objetivo es alcanzar una mayor destreza en la programación con **R**, así como afianzar algunos conceptos claves de estadística básica y empezar a diseñar proyectos de simulación de una manera más estructurada para entender la lógica que subyace.

1 El azar y el pseudo-azar

Entendemos por azar a un proceso cuyo resultado no puede ser predicho completamente, sino solamente en términos probabilísticos. Por ejemplo, tirar una moneda al aire o extraer una bola de un saco (*insaculación*). Si nos preguntásemos por qué ocurre esta indefinición del resultado encontraríamos múltiples respuestas, que nos son ofrecidas desde las matemáticas, la filosofía o incluso la teología. Una de las explicaciones más aceptadas es que el azar es ignorancia, en el sentido que lo plantea el determinismo de *Laplace demonio de Laplace*:

> *Si fuese capaz de conocer el estado de todas las partículas del universo y la fórmula que las rige podría predecir el resultado de tirar una moneda*

Muchos datos parecen avalar esta postura, por ejemplo, procesos que antes eran completamente azarosos (que sea niño o niña, que llueva, etc) hoy no lo son. Y hay muchos casos de sistemas informáticos que pueden predecir qué va a salir en una moneda, en un dado o lo que vas a sacar en piedra, papel o tijera.

Otras posturas filosóficas no defienden ese determinismo, y hay autores que creen que en los procesos cuánticos se produce "verdadero" azar, o por ejemplo, que los fenómenos psicológicos no dependen de procesos deterministas físicos, y por lo tanto son arbitrarios (y existe el libre albedrío). El alumno si quiere que investigue por su cuenta estas cuestiones, pero en cualquier caso, queda claro que la pregunta de si existe realmente el azar enlaza con las cuestiones relevantes de la filosofía sobre Dios, la libertad, etc.

Independientemente de lo anterior, lo que a nosotros nos interesa, y lo que quiero reflejar en este apartado, es que no podemos producir "verdadero azar" en un ordenador. Para ello necesitaríamos algún proceso "físico" y en el ordenador solo contamos con procesos "lógicos". Lo que los ordenadores producen es el llamado "pseudo-azar", que consisten en fórmulas, razonablemente complejas, que producen unos resultados que nos parecen aleatorios, pero que en absoluto lo son. Vamos a verlo con un ejemplo, en R una función aleatoria es por ejemplo *sample()*

```
  sample(1:10, 10)
```

```
##  [1] 10  8  5  6  7  4  2  3  1  9
```

```
  sample(1:10, 10)
```

```
##  [1]  9  8  1  2  5 10  3  4  6  7
```

Le hemos pedido dos veces que genere 10 números aleatorios de una distribución uniforme y me ha generado 20 números que parecen completamente aleatorios, pero mira ahora

```
  set.seed(1)
  sample(1:10, 10)
```

```
##  [1]  9  4  7  1  2  5  3 10  6  8
```

```
  set.seed(1)
  sample(1:10, 10)
```

```
##  [1]  9  4  7  1  2  5  3 10  6  8
```

Las dos series son completamente idénticas, ha generado los mismos números aleatorios. Cuando en un ordenador pedimos una secuencia aleatoria lo que hace es usar esa *fórmula compleja* e iniciarla con una "semilla" (un primer valor), esta semilla, para que parezca que los números son aleatorios, suele ser la hora (en milisegundos), los ciclos de procesador o alguna otra estrategia más compleja. Al cambiar la semilla al invocar la función aleatoria hace que estos sean distintos, pero no porque sean aleatorios, sino porque las semillas han sido distintas. Con la función *set.seed()* le hemos dicho a R que la semilla sea 1 (podríamos haber usado cualquier valor) para las dos secuencias y por eso hemos obtenido la misma serie numérica. Durante el curso siempre que quiera que obtengamos los mismos resultados utilizaré *set.seed*, esto es útil cuando se busca la replicabilidad, es decir, que vosotros al repetir el ejercicio encontréis los mismos resultados que yo al escribir la explicación.

Existe mucha discusión sobre el azar y el pseudo-azar en informática, porque por ejemplo si las claves de seguridad que genera un ordenador no son realmente aleatorias alguien podría destruirlas conociendo la fórmula (y eso ocurre, y por eso la criptografía es tan relevante en computación) pero en principio esas cuestiones no son relevantes para nosotros. Lo que sí nos es relevante es poder demostrar que, aunque solo tengamos pseudo-azar, las simulaciones por ordenador son perfectamente válidas. Principalmente voy a esgrimir dos motivos:

- Sería imposible sacar 1000 millones de veces una bola de un saco, como se hace en cualquier simulación. Aunque no sea la ideal, es la que tenemos.
- Las distribuciones de pseudo-azar se ajustan perfectamente a las distribuciones de azar, es decir, el ordenador produce las mismas distribuciones de probabilidad que obtendríamos con un sistema físico.

2 Generadores aleatorios en R

R puede generar números pertenecientes a cualquier distribución de probabilidad, en general, en el paquete *base* están todas y empiezan por la letra "r", como *rnorm* o *rt* . Las más comunes en estadística son:

2.1 Distribuciones discretas

```
sample(1:10, 10, replace=T)  # Uniforme: 5 numeros del 1 al 10, con reposicion
```

```
##  [1]  3  1  5  5 10  6 10  7  9  5
```

```
set.seed(1)
rbinom(10, 20, 0.5)          # Binomial: 10 repeticiones de un experimento binomial, de 20 ensayos con
```

```
##  [1]  9  9 10 13  8 13 14 11 11  7
```

En la binomial, esos números significan que si tiro una moneda (p=0.5) 20 veces, y ese experimento lo repito 10 veces, la primera vez obtengo 9 exitos (pe. 9 caras), la segunda 9 caras, la tercera 10 caras, etc.

2.2 Distribución continuas

```
runif(10, 1, 10)  # Uniforme: 10 numeros de una uniforme entre 1 y 10
```

```
##  [1] 2.853771 2.589011 7.183206 4.456933 7.928573 5.479293 7.458567 9.927155
##  [9] 4.420317 7.997007
```

```
rnorm(10, 0, 1)   # Normal: 10 numeros de una N(0,1)
```

```
##  [1]  1.51178117  0.38984324 -0.62124058 -2.21469989  1.12493092 -0.04493361
##  [7] -0.01619026  0.94383621  0.82122120  0.59390132
```

```
rt(10, 1)         # t de student: 10 numeros de una t(1)
```

```
##  [1]   1.0361299 -16.8103321  -0.4313783   1.7676553   0.4838340  -2.4255767
##  [7]  -0.5887315   0.5853582  -0.9979839   1.2117985
```

```
rf(10, 1, 1)      # F de Fisher-Snedecor: 10 numeros de una F(1,1)
```

```
##  [1] 2.744272e+00 4.268687e-01 1.088565e+03 4.334863e-02 4.997548e+00
##  [6] 2.616337e+02 1.072740e-01 1.019701e-01 3.635963e+00 2.209186e-01
```

```
rchisq(10, 1)     # Ji-cuadrado: 10 numeros de una X(1)
```

```
##  [1] 0.01141421 1.46837663 1.11414328 2.15372386 0.40586775 2.46808812
##  [7] 1.42015794 0.02852595 0.03414423 0.28342896
```

Las distribuciones t, F, y χ^2 tienen parámetros de centralidad que vamos a obviar, por defecto se generan distribuciones centrales.

Las distribuciones en R no se limitan a las que hemos puesto aquí, hay infinidad de ellas y existe una comunidad muy dinámica que las estudia. Para más información se puede acudir a las CRAN Task View sobre distribuciones.

3 Validación de los generadores de distribuciones en R

En este apartado vamos a comprobar visualmente que los generadores de R funcionan correctamente. Para ello vamos a aprovechar que las funciones de probabilidad (o de densidad de probabilidad) de estas distribuciones están definidas matemáticamente, por ejemplo la función de densidad de probabilidad de la distribución normal es la siguiente:

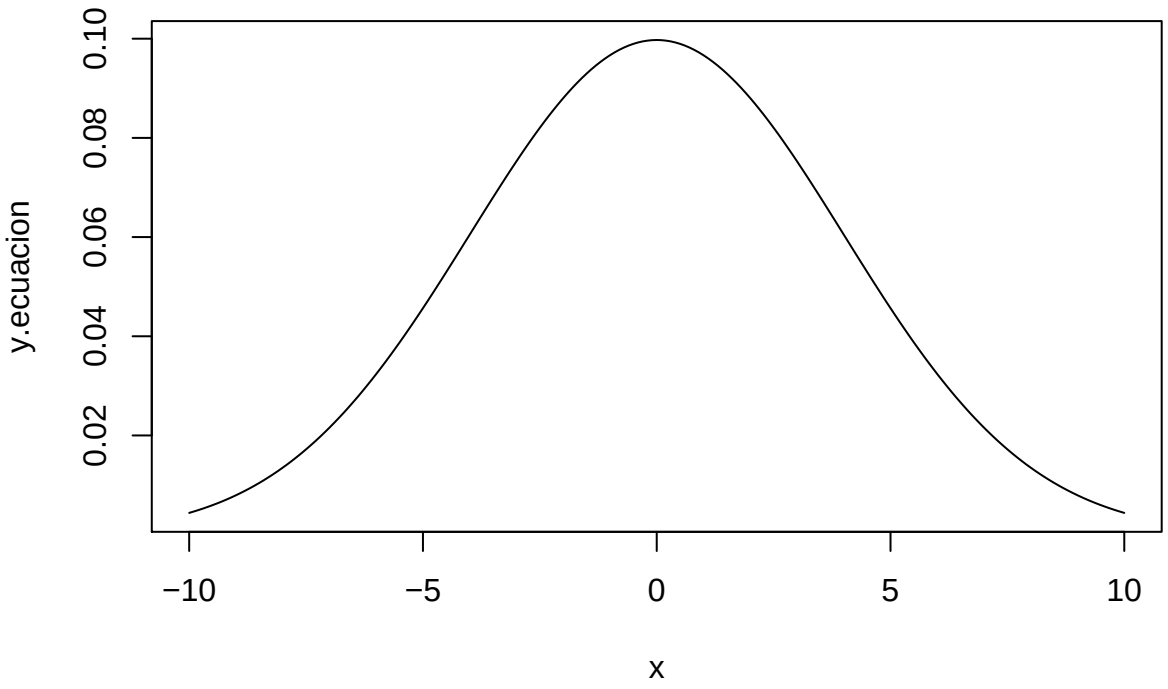
$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/(2\sigma^2)}$$

Podemos construir una función en R para esta ecuación, por ejemplo:

```
normal <- function(x, mu, sigma){
  n = (1/sqrt(2*pi*sigma**2)) * exp(-(x-mu)**2/(2*sigma**2))
}
```

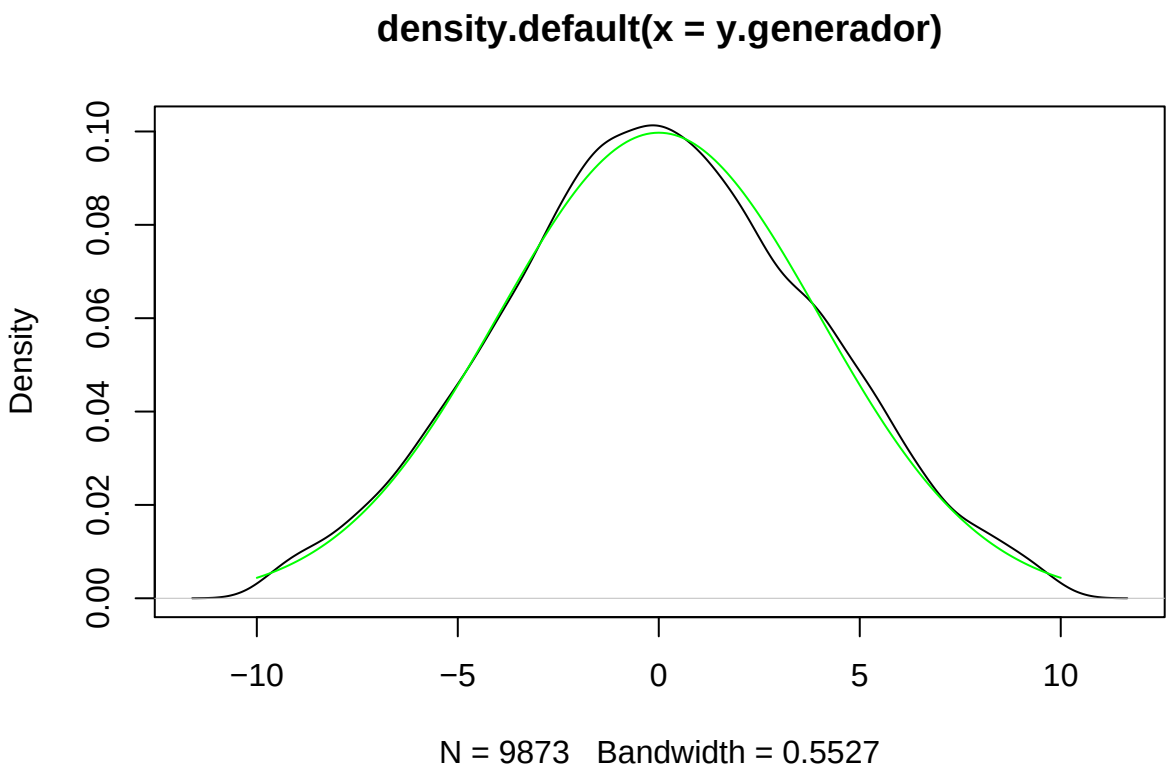
Y ahora podemos usarla para dibujar la función de probabilidad de una variable con media 0 y desviación típica 4, en el intervalo -10 y 10

```
x <- seq(-10, 10, .1)                  # genero los valo  res en x
y.ecuacion <- normal(x, mu=0, sigma=4) # aplico la funcion
plot(y.ecuacion~x, type="l")           # dibujo el resultado
```



Si el generador aleatorio funcionase correctamente, al pedirle a rnorm() que generase números aleatorios con media 0 y desviación típica 4 debería obtener el mismo gráfico. Vamos a comprobarlo:

```
y.generador <- rnorm(10000, 0, 4)  # genero los numeros

# como va a generar numeros entre -inf y +inf puede salir algún valor por debajo de 10 o por encima, as
y.generador <- y.generador[y.generador>-10 & y.generador<10]

d <- density(y.generador)          # para que se vea mejor que un histograma
plot(d)                            # plot d
lines(y.ecuacion~x, col="green")   # superpongo lo obtenido con la formula
```

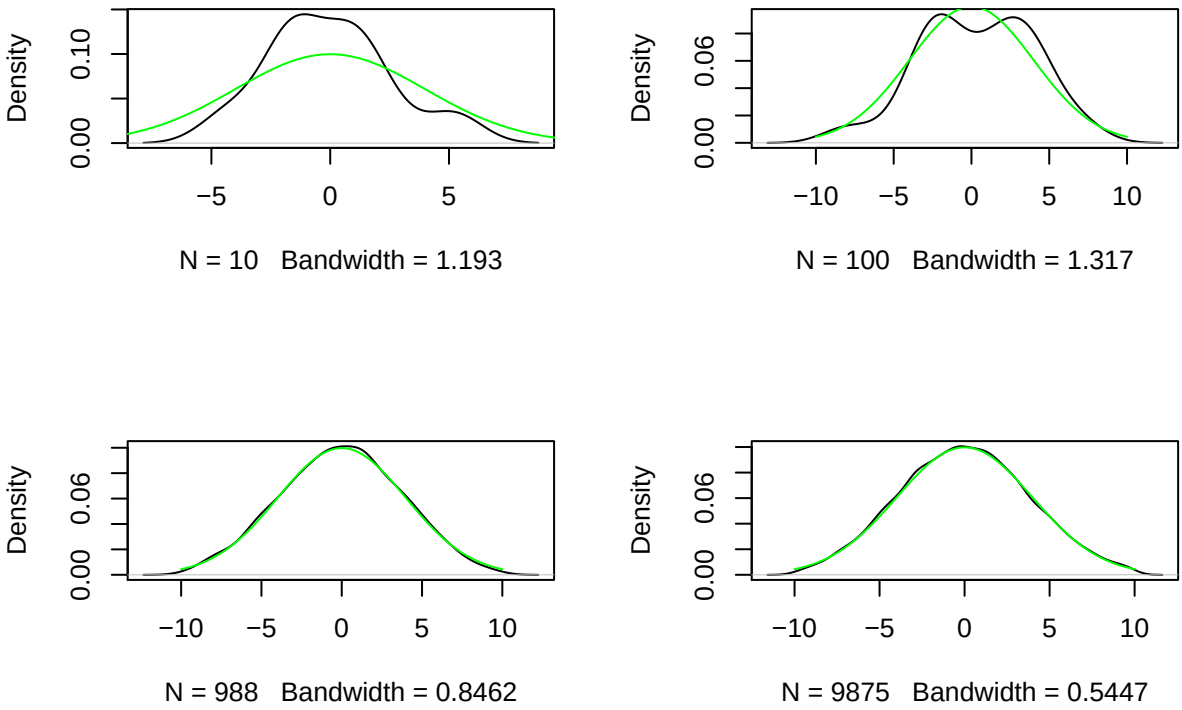


Como se puede comprobar la superposición es casi total entre lo generado por R y la distribución teórica. Y por ese motivo, aunque los generadores no sean estrictamente azar, sino pseudo-azar, podemos utilizarlos para construir simulaciones de datos.

Fíjate en un detalle, mira el siguiente código:

```
# genero numeros, cada vez más muestra
y.generador.10 <- rnorm(10, 0, 4)
y.generador.100 <- rnorm(100, 0, 4)
y.generador.1000 <- rnorm(1000, 0, 4)
y.generador.10000 <- rnorm(10000, 0, 4)

# recortar lo pongo en una funcion para no escribir tanto codigo
recorta <- function(y, li, ls){
  return(y[y>li & y<ls])
}

y.generador.10 <- recorta(y.generador.10,-10,10)
y.generador.100 <- recorta(y.generador.100,-10,10)
y.generador.1000 <- recorta(y.generador.1000,-10,10)
y.generador.10000 <- recorta(y.generador.10000,-10,10)


par(mfrow=c(2,2))

d <- density(y.generador.10)
plot(d, main="")
lines(y.ecuacion~x, col="green")

d <- density(y.generador.100)
```

```r
plot(d, main="")
lines(y.ecuacion~x, col="green")

d <- density(y.generador.1000)
plot(d, main="")
lines(y.ecuacion~x, col="green")

d <- density(y.generador.10000)
plot(d, main="")
lines(y.ecuacion~x, col="green")
```



Al aumentar el tamaño de las muestras, las distribuciones se parecen más y más a la distribución teórica. Es lo que se conoce como la *ley débil de los grandes números*. En un momento y con un poco de código hemos demostrado que la función rnorm() funciona correctamente y que la ley débil se cumple; esta ley, que es una de las leyes fundamentales de la estadística, aparecerá constantemente en todas las simulaciones y la podemos interpretar como: solamente cuando extraigamos muestras numerosas (o repitamos un número grandes de veces el experimento) encontraremos los valores teóricos esperados. Recalco el concepto: **"Solamente tras repetir numerosas veces un experimento nos acercaremos al valor real"**. Cuando hagamos una simulación tendremos que repetirla infinidad de veces hasta que obtengamos una distribución de resultados que podamos interpretar.

3.1 Ejercicios sobre distribuciones

La wikipedia contiene excelente información sobre las funciones de densidad de probabilidad de las distribuciones uniforme, t, F y χ^2.

En estas distribuciones aparecerán dos funciones, denotadas por Γ y β y llamadas funciones Gamma y Beta de Euler y que se definen de la siguiente manera:

$$\Gamma(x) = \int_o^\infty t^{x-1} e^{-t} dt$$

$$\beta(x,y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$$

Pero ninguna de las dos es necesario implementarlas ya que están disponibles en R en las funciones *gamma()* y *beta()*, por ejemplo:

```
gamma(0.5)  # mas informacion usando ?gamma
```

```
## [1] 1.772454
```

```
beta(2,2)   # mas informacion usando ?beta
```

```
## [1] 0.1666667
```

Siguiendo la lógica que he utilizado en el apartado anterior realiza los siguientes ejercicios:

1. Muestra que la función runif produce una distribución uniforme que coincide con la teórica
2. Muestra que rt genera distribuciones t que coinciden con la teórica
3. Muestra que rf genera distribuciones F que coinciden con la teórica
4. Muestra que rchisq genera distribuciones chisq que coinciden con la teórica

Para mostrarlo, simplemente dibuja las distribuciones y comprueba que el histograma se aproxima a su distribución teórica.

Para resolver bien los ejercicios recuerda que tienes que ajustar los valores máximos y mínimos de x para que representen la gráfica. Te será de ayuda conocer la moda (el valor más alto) de cada una de ellas:

$$moda(t) = 0$$

$$moda(F_{gl1,gl2}) = \frac{gl_2}{gl_2 - 2} \text{ para } gl_2 > 2$$

$$moda(\chi^2) = gl - 2 \text{ para } gl > 2$$

4 Sencilla comprobación de la ley débil de los grandes números

Si tiramos una moneda al aire 4 veces no vamos a obtener dos caras y dos cruces, probablemente obtendremos 3 caras y 1 cruz o cualquier otro valor posible. La ley débil de los grandes números establece que solo cuando el número de repeticiones del experimento sea grande (se acerque al infinito) obtendremos el valor esperado real.

Vamos a verlo con R y una binomial. Supongamos que tenemos un experimento que consiste en tirar 10 monedas al aire. De estadística sabemos que el valor esperado será de 10 * 0.5 = 5. Lo que esperamos es que salgan 5 caras de cada experimento.

```
set.seed(26)      # este valor da igual, es solo para que sea el mismo resultado
rbinom(1,10,0.5)  # hago UNA vez el experimento de tirar 10 monedas, cada una con una probabilidad de s
```

```
## [1] 2
```

Ha obtenido un valor de 2, es decir hemos obtenido 2 caras, un poco lejos de las 5 teóricas que deberíamos haber obtenido. Si ahora repetimos el experimento dos veces encontramos:

```
rbinom(2,10,0.5)  # hago DOS veces el experimento de tirar 10 monedas, cada una con una probabilidad de
```

```
## [1] 4 7
```

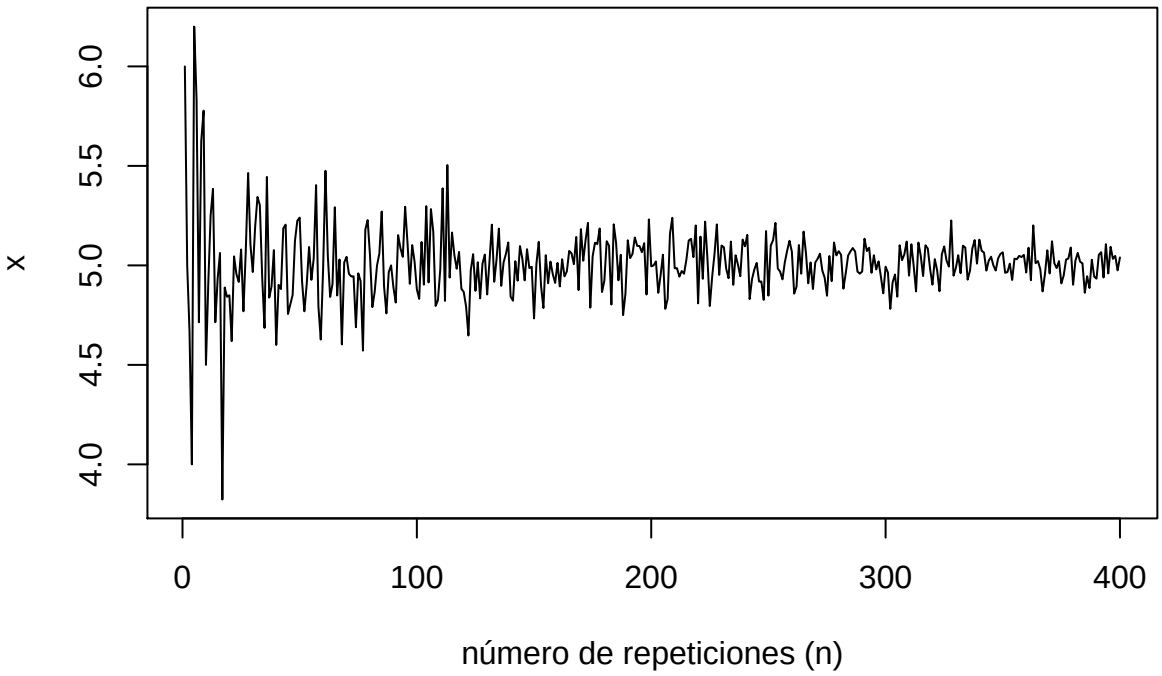
4 y 7, al calcular la media obtengo un valor de 5.5, mucho más cercano al 5 teórico. Ahora lo mismo pero con cuatro repeticiones del experimento:

```r
rbinom(4,10,0.5)  # hago CUATRO veces el experimento de tirar 10 monedas, cada una con una probabilidad
```

```
## [1] 6 4 5 6
```

La media de 6, 4, 5 y 6 nos da una media de 5.25 muy cercana ya. Queda claro que según vamos aumentando las repeticiones del experimento el valor empírico se parece más al teórico. Podemos crear una gráfica en R para visualizar esa aproximación

```r
x <- vector()
for(n in 1:400) x[n] <- mean(rbinom(n,10,0.5))
plot(x, type="l", xlab="número de repeticiones (n)")
```



donde n es el subíndice que representa la cantidad de veces que repito el experimento (en el eje x de la gráfica), puede verse que al aumentar el n el valor obtenido se aproxima al teórico (5) y ambos valores serían idénticos en el infinito.

4.1 Ejercicio sobre la ley débil de los grandes números

5. Muestra como el valor empírico y el teórico se aproximan al aumentar el n en el caso de tirar 5 dados y que salga el 1.

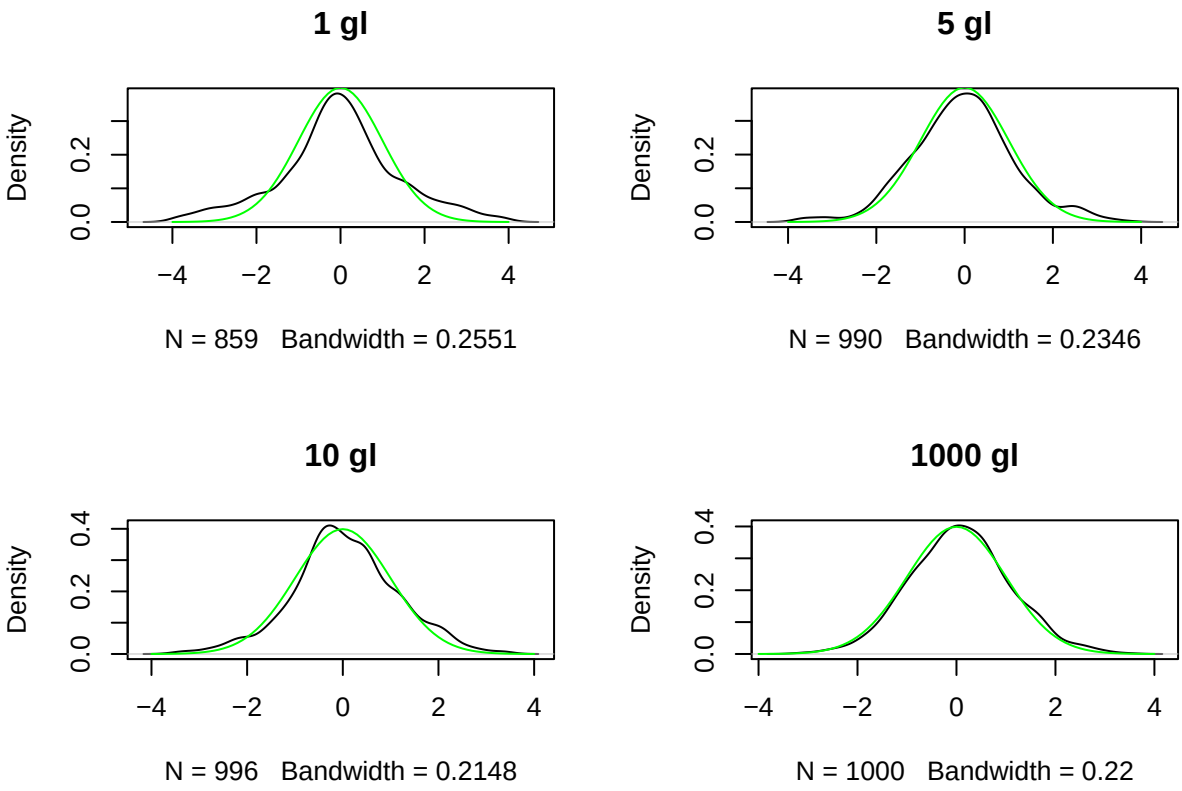
5 Teorema central del límite

Una vez que hemos comprobado que R nos ofrece generadores aleatorios válidos podemos empezar a realizar las primeras simulaciones. La más sencilla, creo, es la del *Teorema central de límite*. Este teorema establece que cualquier distribución acaba por parecerse a la normal si se extrae suficiente muestra. Esto no es exactamente así y supone una gran simplificación, pero nos puede valer para no complicarnos. El tamaño de la muestra en las distribuciones t y F afectan directamente a los grados de libertad, así que podíamos traducirlo como que al aumentar los grados de libertad de una distribución esta se aproxima a la normal. En el caso de χ^2 los grados de libertad suelen depender del número de casillas (como en el contraste de

independencia) pero en definitiva se interpreta igual, al aumentar los grados de libertad se aproximará a la normal.

Veámoslo con R:

```
# las funciones normal y recorta ya estaban definidas
normal <- function(x, med, sd){
  n = (1/sqrt(2*pi*sd**2)) * exp(-(x-med)**2/(2*sd**2))
}

recorta <- function(y, li, ls){
  return(y[y>li & y<ls])
}


x <- seq(-4, 4, .1)             # los valores estaran en este rango
y.normal <- normal(x, 0, 1)     # distribucion teorica normal


y.tgl1 <- rt(1000, 1)          # distribuciones generadas por rt() con gl = 1
y.tgl5 <- rt(1000, 5)          # distribuciones generadas por rt() con gl = 5
y.tgl10 <- rt(1000, 10)        # distribuciones generadas por rt() con gl = 10
y.tgl100 <- rt(1000, 100)      # distribuciones generadas por rt() con gl = 100

# recortamos para simplificar
y.tgl1 <- recorta(y.tgl1, -4, 4)
y.tgl5 <- recorta(y.tgl5, -4, 4)
y.tgl10 <- recorta(y.tgl10, -4, 4)
y.tgl100 <- recorta(y.tgl100, -4, 4)

par(mfrow=c(2,2))
plot(density(y.tgl1), main="1 gl")
lines(y.normal~x, col="green")

plot(density(y.tgl5), main="5 gl")
lines(y.normal~x, col="green")

plot(density(y.tgl10), main="10 gl")
lines(y.normal~x, col="green")

plot(density(y.tgl100), main="1000 gl")
lines(y.normal~x, col="green")
```



Podemos comprobar que al aumentar los grados de libertad (gl) la distribución t se aproxima cada vez más a la normal teórica. En la distribución t prácticamente desde gl=1 las distribuciones son muy parecidas, eso no va a ocurrir con F y χ^2

5.1 Ejercicios sobre el Teorema central del límite

Las figuras anteriores aparecen perfectamente centradas la distribución normal y la t porque ambas tienen la media y la moda en cero, pero esto no ocurre con la distribución F cuya media es: $gl_2/(gl_2 - 2)$ para valores de $gl_2 > 2$ ni con χ^2 cuya media es gl (sus grados de libertad). Recuerde que anteriormente hemos vista la moda de ambas distribuciones, téngalo en cuenta para desplazar las distribución normal lo suficiente y hacer que coincidan en los gráficos.

6. Muestre el teorema central del límite con la distribución F. Como F tiene gl_1 y gl_2, para no hacer infinidad de gráficos, haga la demostración para $gl_1 = gl_2$, es decir, siempre los mismos grados de libertad en el numerador y el denominador.
7. Muestre el teorema central del límite con la distribución χ^2.