Tutorial 12: Contrastes χ^2

Atención:

- Este documento pdf lleva adjuntos algunos de los ficheros de datos necesarios. Y está pensado para trabajar con él directamente en tu ordenador. Al usarlo en la pantalla, si es necesario, puedes aumentar alguna de las figuras para ver los detalles. Antes de imprimirlo, piensa si es necesario. Los árboles y nosotros te lo agradeceremos.
- Fecha: 19 de abril de 2017. Si este fichero tiene más de un año, puede resultar obsoleto. Busca si existe una versión más reciente.

Índice

1. Contrastes χ^2 (independencia y homogeneidad) con R

En esta sección vamos a utilizar R para realizar un contraste de independencia, como el del ejemplo del libro sobre la posible relación entre el género y las creencias religiosas, basado en datos del *Barómetro* del CIS (Ejemplo 12.1.1, pág. 466. Recuerda que en ese ejemplo nos preguntábamos si la proporción de creyentes es distinta entre hombres y mujeres. O como el del ejemplo sobre la composición por género de poblaciones de Avutardas (Ejemplo 12.1.5, pág. 473) en el que nos preguntamos si la proporción de machos, hembras y juveniles varía de unas poblaciones a otras.

Antes de empezar, queremos recordar la última de las observaciones de la página 468 del libro. Aunque el Ejemplo 12.1.1 del *Barómetro* empieza con una tabla incompleta, que sólo contiene los valores marginales, en una aplicación típica de este método empezamos con los valores observados y, a partir de ellos, calculamos los esperados. Eso es lo que vamos a hacer aquí, tomar los valores observados como punto de partida.

Con estas premisas, podemos empezar a centrar el problema. Vamos a suponer que queremos contrastar la posible relación $F_1 \sim F_2$ entre dos factores F_1 y F_2, con n_1 y n_2 niveles, respectivamente. Lo haremos basándonos en una tabla de contingencia de valores observados o_{ij}, de dimensiones $n_1 \times n_2$, como la parte central (sin los márgenes) de la Tabla 12.1.2 del libro (pág. 472), que reproducimos aquí:

	Factor F_2		
Factor F_1	o_{11}	$\cdots$	o_{1n_2}
		$\ddots$	
	$o_{n_2 1}$	$\cdots$	$o_{n_1 n_2}$

1.1. El test de independencia paso a paso

Vamos a hacer, paso a paso, los cálculos necesarios para obtener el contraste χ^2 de independencia para el Ejemplo 12.1.1, el del *Barómetro* del CIS.

Tabla de valores observados.

El punto de partida es la tabla de valores observados. Vamos a suponer que esa tabla está almacenada en un objeto llamado `tablaObservada`, de tipo `matrix` (ver el Tutorial04) o posiblemente en un `data.frame`. En el trabajo que vamos a hacer aquí, no hay mucha diferencia entre usar uno u otro objeto. Para el Ejemplo 12.1.1 del *Barómetro* del CIS, podemos crear ese objeto como una matriz mediante este comando:

```
(tablaObservada = matrix( c(849, 1015, 356, 232),  nrow= 2, byrow = TRUE))
```

```
##      [,1] [,2]
## [1,]  849 1015
## [2,]  356  232
```

Lo primero que vamos a hacer, para ayudarnos en la discusión, es calcular las dimensiones de esta matriz:

```
(nFilas = nrow(tablaObservada))
```

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

```
(nColumnas = ncol(tablaObservada))
```

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

y también el número total de observaciones:

```
(n = sum(tablaObservada) )
```

```
## [1] 2452
```

A continuación vamos a *decorar* esta matriz, cambiando los nombres de filas y columnas para que nos recuerden a qué nivel del correspondiente factor nos estamos refiriendo. En este caso vamos a usar unos nombres que nos recuerden el significado de los datos que estamos manejando:

```
colnames(tablaObservada) = c("H", "M")
rownames(tablaObservada) = c("CREE", "NO_CREE" )
tablaObservada
```

```
##           H    M
## CREE    849 1015
## NO_CREE 356  232
```

Si el número de niveles es elevado, tal vez prefieras que R se encargue de poner nombre de forma automática a las filas y columnas. En el fichero plantilla encontrarás unas líneas de código que se encargan precisamente de esto, y que usan la función `paste` para conseguirlo.

El siguiente paso es calcular los valores marginales. Para ello disponemos en R de la función `addmargins`. Vamos a guardar el resultado en otra matriz, que llamaremos `tablaObservadaMarg`, para, por un lado, poder acceder fácilmente a esos valores marginales, pero a la vez evitando modificar la tabla observada original. Además, vamos a usar una función parecida, llamada `margin.table`, para guardar los valores marginales en dos vectores, que usaremos más adelante.

```r
(tablaObservadaMarg = addmargins(tablaObservada))
```

```
##            H    M  Sum
## CREE     849 1015 1864
## NO_CREE  356  232  588
## Sum     1205 1247 2452
```

```r
(marginalesFilas = margin.table(tablaObservada, margin=1) )
```

```
##    CREE NO_CREE
##    1864     588
```

```r
(marginalesColumnas = margin.table(tablaObservada, margin=2) )
```

```
##    H    M
## 1205 1247
```

Fíjate en que, en la función `margin.table`, usamos la opción `margin = 1` para filas, y la opción `margin = 2` para columnas.

Ejercicio 1. *Lee la ayuda de la función* `addmargins`*, para ver que permite hacer más cosas de las que hemos mostrado aquí.* □

Tabla de valores esperados.

¿Cómo podemos fabricar la tabla de valores esperados a partir de estos dos vectores? Recuerda que la tabla de valores esperados se calcula usando la Ecuación 12.5 (472) del libro, que dice:

$$e_{ij} = \frac{o_{i+} \cdot o_{+j}}{o_{++}}.$$

Desde el punto de vista matemático, el numerador de esta fórmula describe el producto matricial de los dos vectores de sumas marginales. Si no recuerdas o no sabes cómo funciona el producto de matrices (¡conviene que lo aprendas, más pronto que tarde lo necesitarás!), puedes limitarte a aplicar el resultado que vamos a ver. Para saltar hasta ese punto, busca el siguiente frailecillo, como el que aparece en el margen.

Pero para los lectores que sí sepan como funciona ese tipo de productos, el vector o_{filas}, de sumas marginales por filas, es un vector fila, de dimensiones 2×1, mientras que el vector $o_{columnas}$, de sumas marginales por columnas es un vector fila, de dimensiones 1×2. Así que el producto matricial

$$o_{filas} \cdot o_{columnas}$$

da como resultado la matriz 2×2 de valores esperados. Esa es la visión matricial de la Ecuación 12.5 del libro.

Y ahora, para aplicar esto a nuestro problema necesitaremos recordar cómo se hace un producto matricial en R (lo vimos en la Sección 2.6 del Tutorial03, pág. 14). Primero empezamos por convertir el objeto `marginalesFilas` de tipo `vector` en un objeto de tipo `matrix`. Podemos conseguir esto simplemente cambiando sus dimensiones, con lo que estremos listos para calcular el producto matricial con `%*%`:

```r
dim(marginalesFilas)=c(nFilas, 1)
tablaEsperada = (marginalesFilas %*% marginalesColumnas) / n
```

Antes de mostrar el resultado vamos a usar los mismos nombres de filas y columnas que usamos en la matriz observada:

```
colnames(tablaEsperada)=colnames(tablaObservada)
rownames(tablaEsperada)=rownames(tablaObservada)
```

Finalmente añadimos los valores marginales de esta tabla esperada y la mostramos. Los valores marginales deben coincidir con los de la tabla observada (salvo quizá por el redondeo en algunos casos).

El resumen final, en cualquier caso, es que hemos obtenido esta tabla de valores esperados:

```
tablaEsperada
```

```
##               H      M
## CREE     916.04 947.96
## NO_CREE 288.96 299.04
```

Comprueba que estos valores son (salvo el redondeo), los que aparecen en el Ejemplo 12.1.1 del libro. No vamos a redondear estos valores, porque eso afectaría al p-valor y haría que nuestros resultados fueran distintos de los que calcula R directamente.

Estadístico del contraste χ^2 y cálculo del p-valor.

Una vez que disponemos de las dos matrices, las cuentas del contraste de independencia son muy sencillas. El estadístico Ξ, de la Ecuación 12.3 (pág. 469), que es

$$\Xi = \frac{(o_{11} - e_{11})^2}{e_{11}} + \frac{(o_{12} - e_{12})^2}{e_{12}} + \frac{(o_{21} - e_{21})^2}{e_{21}} + \frac{(o_{22} - e_{22})^2}{e_{22}}$$

se calcula en R con una sola línea de código (se muestra la salida):

```
(Estadistico = sum((tablaObservada - tablaEsperada)^2 / tablaEsperada))
```

```
## [1] 40.225
```

A partir de este resultado el p-valor es inmediato:

```
(pValor = 1 - pchisq(Estadistico, df=(nFilas - 1) * (nColumnas - 1)))
```

```
## [1] 2.263e-10
```

Como ves, hemos usado la opción `correct=FALSE`. El efecto es similar al que hemos visto en otras ocasiones en el libro: le pedimos a R que no use correciones de continuidad y, de hecho, que no obtenga "el mejor resultado posible", para que la respuesta coincida con nuestros cálculos elementales. En una aplicación a un problema del mundo real, desde luego usaríamos `correct=TRUE`.

La función `chisq.test` para el cálculo directo.

Para no tener que hacer todas esas operaciones a mano cada vez, en R disponemos de la función `chisq.test`, que permite obtener el estadístico del contraste, los grados de libertad y el p-valor de forma muy sencilla. En nuestro ejemplo bastaría con hacer:

```
(chisqTest = chisq.test(tablaObservada, correct=FALSE))
```

```
##
##  Pearson's Chi-squared test
##
## data:  tablaObservada
## X-squared = 40.2, df = 1, p-value = 2.3e-10
```

Como ves, la salida incluye el valor del estadístico (que en el libro hemos llamado Ξ), el número de grados de libertad y el p-valor del contraste. Hemos guardado el resultado en la variable `chisqTest`, porque de esa forma podemos acceder a información adicional usando la construcción con `chisqTest$` que hemos visto en otros casos. Por ejemplo, la matriz esperada se obtiene de esta forma tan simple:

```
chisqTest$expected
```

```
##              H      M
## CREE    916.04 947.96
## NO_CREE 288.96 299.04
```

Pero hay más información disponible, muy útil para un análisis más profundo del contraste χ^2 de independencia (un análisis que no hemos hecho en el libro). Si el contraste es positivo, tenemos evidencia para creer que existe una relación de dependencia entre los dos factores F_1 y F_2 que intervienen en el contraste. Pero que exista una dependencia no nos dice gran cosa sobre la *fuerza* de esa relación. En particular, por pequeño que sea el p-valor que hayamos obtenido, seguimos sin saber si la relación es fuerte o no. ¿Cómo podríamos medir la intensidad de la relación? Pues por ejemplo, puedes usar la salida de `chisq.test` para obtener los residuos, y los residuos estandarizados del contraste. Los residuos, a secas, son simplemente las diferencias

$$o_{ij} - e_{ij}$$

entre los valores esperados y los observados.

```
chisqTest$residuals
```

```
##               H       M
## CREE    -2.2149  2.1773
## NO_CREE  3.9435 -3.8766
```

Pero, puesto que el tamaño de esas diferencias depende, por ejemplo, del tamaño de la muestra, no es una buena idea usar el tamaño de los residuos, sin más, para medir la fuerza de la relación entre F_1 y F_2. Para eso se usan los residuos estandarizados, que son una especie de tipificación de los residuos, para llevarlos a una escala normal estándar donde poder medirlos adecuadamente.

```
chisqTest$stdres
```

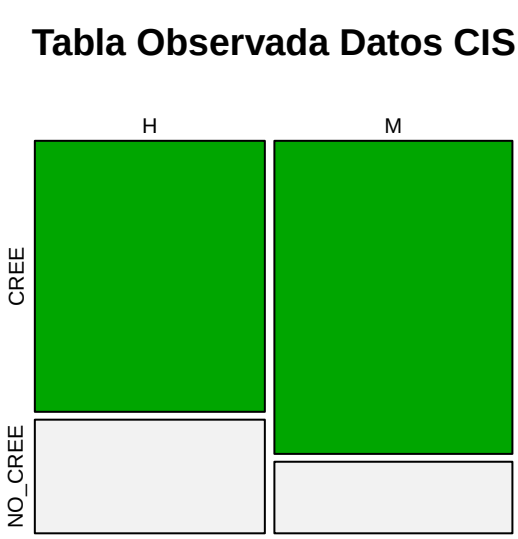
```
##               H       M
## CREE    -6.3423  6.3423
## NO_CREE  6.3423 -6.3423
```

No queremos, en este tutorial, extendernos mucho más en la discusión. Una referencia básica para este tipo de análisis es el libro *Categorical Data Analysis, 3rd Edition*, de Alan Agresti, publicado en Wiley (ISBN: 978-1-118-71094-4).

Representación gráfica de una tabla de contingencia. El gráfico de mosaico.

En el segundo ejemplo de contraste de independencia del libro, el Ejemplo 12.1.5 de las poblaciones de avutardas, hemos usado un tipo especial de gráfico, el llamado *gráfico de mosaico* para ilustrar los datos de una tabla de contingencia (ver la la Figura 12.2 (pág. 477) del libro). En este ejemplo, ese gráfico se obtiene así:

```
mosaicplot(t(tablaObservada), col=terrain.colors(nColumnas), main="Tabla Observada Datos CIS")
```



Como ves, al tratarse de factores con sólo dos niveles, es una representación muy sencilla. Hemos traspuesto la tabla para que filas y columnas coincidan con la forma en que hemos presentado la tabla anteriormente. La altura y anchura relativa de las columnas nos informa de la proporción relativa de los dos niveles para cada uno de los factores (género en columnas, creencias religiosas en filas).

1.2. Fichero de código R para el contraste χ^2 de independencia.

Los pasos que hemos ido dando para ilustrar en concreto el Ejemplo 12.1.1 del libro se generalizan fácilmente a otros casos similares. Es bueno, como hemos hecho en otras ocasiones, tener preparado un *fichero plantilla* de código R en el que se automaticen al máximo estos pasos, por comodidad de uso y para evitarnos errores. El código que resume todo el trabajo de la sección anterior aparece en el fichero plantilla:

Tut12-Chi2Independencia.R

El fichero permite obtener este tipo de contrastes, paso a paso, y te sugerimos que lo uses para acompañar la discusión de esos ejemplos del libro. Como siempre, conviene que leas primero ese fichero y te familiarices con su funcionamiento en ejemplos sencillos, antes de intentar usarlo en algún otro caso más complicado o importante. En particular, como verás, ese fichero permite comenzar a partir de una tabla de valores observados descrita de varias formas. Una de esas formas es leyendo los datos a partir de un fichero `csv`. Para darte ocasión de practicar con el fichero, aquí tienes varios ejercicios.

Ejercicio 2.

1. *Utiliza ese fichero plantilla para comprobar las cuentas del Ejemplo 12.1.5 del libro (pág. 473), el de las poblaciones de Avutardas. Introduce los datos de la Tabla 12.5 del libro (pág. 474) por filas y por columnas.*
2. *En el fichero Tut12-Avutardas.csv tienes esos mismos datos, para que practiques la lectura de una tabla de contingencia a partir de un fichero* `csv`.

□

1.3. El contraste de homogeneidad en R.

La función `chisq.test` que hemos visto antes es la forma más sencilla de hacer un contraste de homogeneidad en R. Vamos a ver cómo usar esa función para hacer dos ejemplos de la Sección 12.2 del libro: el Ejemplo 12.2.1 del dado cargado (pág. 480), y el Ejemplo 12.2.4 sobre el trabajo de G. Mendel (pág. 483). La razón para hacer los dos es, por supuesto, que el primero de ellos cubre el caso en el que la distribución de probabilidad esperada es equiprobable, mientras que en el segundo caso no lo es.

El ejemplo del dado cargado.

Para el primero de esos dos ejemplos, tenemos un vector de frecuencias observadas:

```
Observadas = c(811, 805, 869, 927, 772, 816)
```

En este caso, los seis posibles valores del dado serían equiprobables *si la hipótesis nula del contraste* χ^2 *fuese cierta.* Para que no quede duda, esa hipótesis nula dice:

$$H_0 = \{\text{el dado no está cargado}\} = \left\{\text{la probabilidad de cada uno de los valores es } \frac{1}{6}\right\}$$

Ese caso equiprobable es el que R asume por defecto, si no le proporcionamos más valores que los observados, aunque nosotros vamos a escribir las probabilidades para hacerlas explícitas. Así que, para realizar el contraste χ^2 en este caso, basta con este comando tan sencillo:

```
(ChisqTest = chisq.test(Observadas, p=rep(1, 6)/6))

##
##  Chi-squared test for given probabilities
##
## data:  Observadas
## X-squared = 18.5, df = 5, p-value = 0.0024
```

Como ves, el estadístico y el p-valor son los que hemos descrito en el Ejemplo 12.2.1 del libro. De nuevo, hemos usado una variable (en este caso `ChisqTest`) para almacenar el resultado del contraste, porque así podemos usar `$` para acceder a otros aspectos del contraste que R no muestra por defecto en la salida de la función `chisq.test`. Por ejemplo, podemos usar este método para obtener los valores esperados que R calcula usando la hipótesis (nula) de equiprobabilidad. Se obtienen así:

```
ChisqTest$expected

## [1] 833.33 833.33 833.33 833.33 833.33 833.33
```

y son, como hemos visto en el libro, el resultado de dividir entre 6 el número total de observaciones, puesto que en este ejemplo hay seis valores posibles.

Los guisantes de Mendel.

En el Ejemplo 12.2.4 del libro, sobre el trabajo de G. Mendel con guisantes, tenemos un vector de frecuencias observadas (semilla lisa, semilla rugosa):

```
Observados = c(5474, 1850)
```

pero ahora, a diferencia del caso anterior, también tenemos un vector de probabilidades esperadas, que son

```
probEsperados = c(3/4, 1/4)
```

Con estos ingredientes, R no necesita nada más para llevar a cabo el contraste χ^2 de homogeneidad. Hacemos simplemente (se muestra la salida):

```
(ChisqTest = chisq.test(Observados, p = probEsperados))

##
##  Chi-squared test for given probabilities
##
## data:  Observados
## X-squared = 0.263, df = 1, p-value = 0.61
```

Y obtenemos el valor del estadístico y el p-valor que hemos visto en el Ejemplo 12.2.4 del libro. Ten en cuenta que, en este caso, es muy importante incluir el nombre `p=` al usar el argumento de las probabilidades, para que R entienda correctamente que lo que queremos hacer es un contraste de homogeneidad.

Ejercicio 3. *Prueba a ejecutar el comando sin ese nombre. Es decir, ejecuta:*

```
chisq.test(Observadas, probEsperadas)
```

y observa lo que sucede. □

1.4. Tablas de contingencia relativas en R.

Vamos a ver cómo utilizar R para obtener las tablas relativas que hemos discutido en la página 478 del libro. Concretamente, vamos a ver cómo reproducir los resultados del Ejemplo 12.1.6, en el que se analizaba la tabla de contingencia correpsondiente a una prueba diagnóstica, que hemos usado varias veces en el libro. Empezamos con la tabla de datos básica :

```
(tablaObservada = matrix( c(192, 4, 158, 9646),  nrow= 2))

##      [,1] [,2]
## [1,]  192  158
## [2,]    4 9646
```

Ponemos nombre a las filas y columnas:

```
colnames(tablaObservada) = c("Enfermos", "Sanos")
rownames(tablaObservada) = c("Positivo", "Negativo" )
tablaObservada

##          Enfermos Sanos
## Positivo      192   158
## Negativo        4  9646
```

y ya estamos listos para pasar a los valores marginales. Los añadimos a la tabla pero, además, calculamos la suma total:

```
(tablaObservadaMarg = addmargins(tablaObservada))

##          Enfermos Sanos   Sum
## Positivo      192   158   350
## Negativo        4  9646  9650
## Sum           196  9804 10000

(n = sum(tablaObservada) )

## [1] 10000
```

Una primera forma de proceder es dividir toda la tabla por n

```
(tablaRelTotales = tablaObservadaMarg / n)
```

```
##          Enfermos  Sanos   Sum
## Positivo   0.0192 0.0158 0.035
## Negativo   0.0004 0.9646 0.965
## Sum        0.0196 0.9804 1.000
```

Cuando lo que queremos es dividir cada fila por la suma total de los elementos de esa fila podemos usar la función `prop.table`, indicando con `margin=1` que queremos usar las filas:

```
(tablaMarginalFilas = addmargins(prop.table(tablaObservada, margin = 1)))
```

```
##            Enfermos   Sanos Sum
## Positivo 0.54857143 0.45143   1
## Negativo 0.00041451 0.99959   1
## Sum      0.54898594 1.45101   2
```

Hemos añadido los márgenes para hacer más evidente la estructura de valores de la tabla. De esa forma queda claro que esta tabla está construida de manera que las sumas totales *por filas* sean 1. Es aconsejable hacer esto, especialmente en tablas más grandes, para evitar confusiones e interpretaciones erróneas de esas tablas.

Y si queremos usar las columnas:

```
(tablaMarginalColumnas = addmargins(prop.table(tablaObservada, margin = 2)))
```

```
##          Enfermos    Sanos     Sum
## Positivo 0.979592 0.016116 0.99571
## Negativo 0.020408 0.983884 1.00429
## Sum      1.000000 1.000000 2.00000
```

2. Datos *en bruto* y *datos limpios* para χ^2.

A lo largo del curso hemos distinguido entre problemas reales (con datos *en bruto*) y los que llamamos *problemas de libro*. En particular, como el lector ya habrá adivinado, entre la recogida de los datos y los vectores, tablas y ficheros que hemos utilizado en ejemplos y ejercicios hay un trabajo intermedio que resulta imprescindible para que podamos aplicar los procedimientos que hemos estudiado hasta ahora.

Veamos un caso concreto. La matriz del Ejemplo 12.1.1, el del *Barómetro*, tiene cuatro elementos, pero representa el resumen de un conjunto de 2452 datos *en bruto*. Cada uno de esos datos en bruto es una observación individual de los dos factores F_1 y F_2, como por ejemplo:

(mujer, no creyente)

Y en la matriz de datos observados hemos resumido 2452 datos como este en tan sólo cuatro números, que llamamos *datos resumidos*. Esas matrices de *recuentos* son resúmenes estadísticos, similares a las tablas de frecuencia, las medias muestrales, etc. En esta sección vamos a aprender algunas técnicas que nos permiten pasar de los datos en bruto a la tabla de valores observados.

Al hacer esto a veces nos encontramos con un problema adicional. En el ejemplo del *Barómetro* ha sido suficiente con hacer un recuento del número de individuos que detenta cada combinación de dos niveles (uno de cada factor) porque ambas variables son *cualitativas*. Pero en otros casos puede que las variables iniciales sean cuantitativas y debamos convertirlas en factores agrupando por clases. En esta sección vamos a empezar con un ejemplo de esta situación.

La pregunta a la que vamos a tratar de responder es si hay diferencia entre los diámetros de los cráteres entre ambos hemisferios de la Luna. Usaremos datos del *Lunar Orbiter Laser Altimeter instrument (LOLA)*, que ya mencionamos en el Ejemplo 9.2.1. Allí incluíamos un fichero `csv`, que reproducimos aquí:

Cap09-LolaLargeLunarCraterCatalog.csv

Las tres variables que aparecen en ese fichero:

```
Lon, Lat, Diam_km
```

se refieren a la latitud, longitud (ambas en grados) y diámetro (en km) de los cráteres lunares y son todas ellas cuantitativas continuas.

```
crateres = read.table(file="../datos/Cap09-LolaLargeLunarCraterCatalog.csv",
                      header=TRUE, sep=",")
```

Podemos determinar a qué hemisferio pertenece un cráter simplemente viendo si su latitud es positiva o negativa. Para hacer esto vamos a agrupar los valores de la variable `lat` (latitud) en dos clases,

$$(-90, 0], (0, 90]$$

que indican simplemente si el cráter se encuentra situado en el hemisferio norte o en el sur. El factor resultante se llama `hemisphere`. En R, como sabemos, la herramienta para hacer este tipo de operaciones es la función `cut`, que en este caso funciona así:

```
hemisphere = cut(crateres$Lat, breaks=c(-90, 0, 90))
head(hemisphere, 20)

##  [1] (-90,0] (-90,0] (0,90]  (0,90]  (0,90]  (-90,0] (-90,0] (-90,0]
##  [9] (-90,0] (-90,0] (-90,0] (-90,0] (-90,0] (-90,0] (0,90]  (0,90]
## [17] (0,90]  (0,90]  (0,90]  (0,90]
## Levels: (-90,0] (0,90]
```

Para mejorar la legibilidad de los datos, vamos a cambiar las etiquetas de los factores:

```
levels(hemisphere) = c("SUR", "NORTE")
head(hemisphere, 20)

##  [1] SUR   SUR   NORTE NORTE NORTE SUR   SUR   SUR   SUR   SUR
## [12] SUR   SUR   SUR   NORTE NORTE NORTE NORTE NORTE NORTE
## Levels: SUR NORTE
```

Y ahora podemos hacer una tabla de frecuencias de esta variable:

```
table(hemisphere)

## hemisphere
##   SUR NORTE
##  2783  2402
```

Por su parte, la variable `Diam_km`, correspondiente al diámetro, tiene un rango muy amplio, que va desde poco más de 20km hasta más de 2000km, y está muy sesgada a la derecha, como puedes ver en su boxplot, que aparece en la Figura 1.

Para apreciar con más claridad la forma de la distribución, en la Figura 2 tienes de nuevo el boxplot, pero eliminando los valores atípicos (se consigue, en R, con la opción `outline=FALSE`).

Ahora que hemos hecho la exploración inicial de la variable `craterSize` podemos pensar cuál es la mejor forma de agruparla en clases. Cuando se agrupan los datos, hay dos alternativas básicas: usar intervalos de la misma anchura, o dividirlos en intervalos que tengan algún sentido en el contexto del problema. En este caso, a la vista de los diagramas anteriores, hemos optado por dividirla en los siguientes cuatro intervalos (en km):

$$[20, 40], (40, 60], (60, 80], [80,)$$

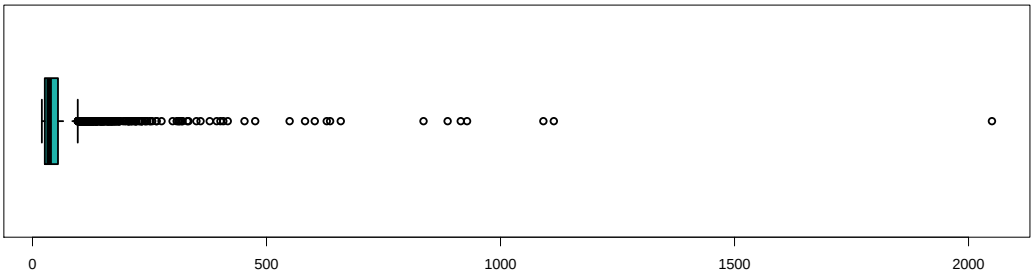


Figura 1: Boxplot, incluyendo valores atípicos, de la variable `Diam_km` del fichero `Cap09-LolaLargeLunarCraterCatalog.csv`.

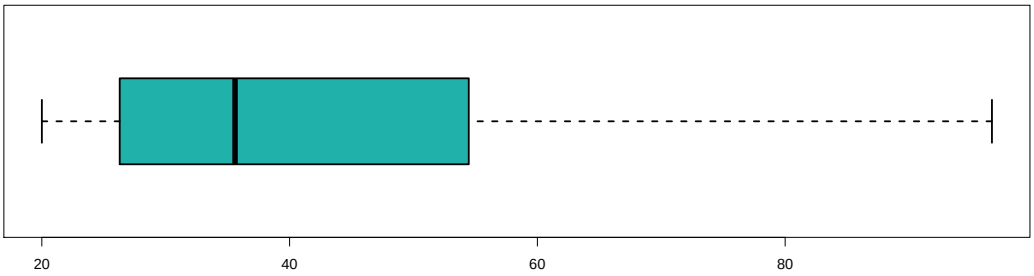


Figura 2: Boxplot, ahora sin los valores atípicos, de la variable `Diam_km` del fichero `Cap09-LolaLargeLunarCraterCatalog.csv`.

donde el último intervalo agrupa a todos los cráteres con un radio superior a 80km (y por tanto, a todos los atípicos de la Figura 1).

```
craterSize = cut(crateres$Diam_km,
                 breaks=c(seq(20, 80, 20), max(crateres$Diam_km)),
                 include.lowest=TRUE)
```

El factor `craterSize` resultante tiene esta tabla de frecuencias:

```
table(craterSize)

## craterSize
##       [20,40]       (40,60]       (60,80] (80,2.05e+03]
##          3003          1108           482           592
```

Recuerda que la pregunta a la que estamos tratando de responder es si existe alguna relación entre el tamaño de los cráteres y su posición en uno u otro hemisferio. Es decir, usando los factores que acabamos de construir, si existe alguna relación entre `hemisphere` y `craterSize`. Para responder a la pregunta vamos a formularla como un contraste χ^2 de independencia entre estos dos factores. La hipótesis nula de ese contraste sería:

$$H_0 = \{\texttt{craterSize} \text{ es independiente de } \texttt{hemisphere}\}.$$

Usaremos este ejemplo para ilustrar cómo podría ser un fichero de datos en bruto para este tipo de problemas y también veremos qué significa que los datos sean limpios en este contexto. Para empezar, vamos a trabajar con el fichero

Tut12-CrateresLunaEnBruto-01.csv

Para ilustrar el contenido de ese fichero, aquí tienes sus primeras 10 filas (el fichero tiene más de 5000 filas de datos).

```
##    SUR NORTE X.20.40. X.40.60. X.60.80. X.80
## 1    1     0        0        0        1    0
## 2    1     0        1        0        0    0
## 3    0     1        0        0        1    0
## 4    0     1        1        0        0    0
## 5    0     1        0        1        0    0
## 6    1     0        0        1        0    0
## 7    1     0        0        0        0    1
## 8    1     0        1        0        0    0
## 9    1     0        0        1        0    0
## 10   1     0        1        0        0    0
```

Los datos que aparecen en este fichero corresponden a los niveles de los factores `hemisphere` y `craterSize`. Un 1 en una columna significa que la observación corresponde a ese nivel del factor (y un 0 que no corresponde). Hemos elegido este formato para el fichero porque:

- Este tipo de representación de los datos se encuentra a veces (especialmente en encuestas), cuando la persona que recoge los datos tiene un formulario con los campos predefinidos, con casillas correspondientes a las distintas clases, y tiene que marcar una de esas casillas (como sucede con los formularios OMR para lectoras ópticas, ver

  http://en.wikipedia.org/wiki/Optical_mark_recognition)

  Como puedes ver, en cada línea del fichero se describe un cráter. Las dos primeras columnas se han usado para indicar el hemisferio en el que se encuentra, mientras que las cuatro últimas indican a cuál de los cuatro niveles del factor `craterSize` pertenece ese cráter.

- Por esa misma razón, no son datos limpios. Aunque cada fila corresponde a una observación individual, no se cumple que cada columna corresponda una variable.

- Además, entre ambos ficheros, se cubren las formas habituales de recoger los *datos de campo*. Date cuenta de que el diseño de la encuesta condiciona el formato en el que recibirás los datos y, por tanto, el esfuerzo que habrá que dedicar a esquilarlos. Este proceso de recogida y formateo de los datos es también parte del Diseño Experimental. Una parte del trabajo que a menudo se obvia en las discusiones teóricas, pero que es esencial porque condiciona todo lo que podemos hacer después con los datos. En particular, fíjate en que puedes pasar del fichero `Cap09-LolaLargeLunarCraterCatalog.csv` al `Tut12-CrateresLunaEnBruto-01.csv`, pero no al revés porque al agrupar los datos se ha producido pérdida de información. Por eso es esencial tener claros los objetivos que perseguimos, para no desperdiciar recursos en conseguir datos innecesarios, a la vez que conservamos con mucho cuidado la información que, de hecho, hemos obtenido.

Aunque hemos fabricado este ejemplo con el propósito de ilustrar la discusión de esta Sección, puedes encontrar muchos otros, empezando por los que se incluyen en el artículo de H. Wickham citado en el Tutorial11. El ejemplo resulta algo artificial porque disponiendo de los datos sin agrupar, lo razonable es obtener un fichero limpio de datos agrupados. Pero le pedimos indulgencia al lector; en el mundo real, en el que a menudo tenemos que trabajar con datos recogidos por otras personas, o por nosotros mismos si no hemos sido suficientemente cuidadosos, abundan los casos con ficheros como éste.

Usando la terminología de ese Tutorial11, ¿cómo podemos *esquilar* el fichero de datos para conseguir lo que queremos? En un ejemplo como este es relativamente sencillo, porque lo único que tenemos que hacer es *fusionar* todas las columnas que corresponden a una misma variable. En concreto, queremos pasar de esto:

```
"SUR","NORTE","[20,40]","(40,60]","(60,80]",">80"
1,0,0,0,1,0
0,1,1,0,0,0
```

a esto (la primera línea es la cabecera de los datos limpios):

```
"hemisphere", "craterSize"
"SUR", "(60,80]"
"NORTE", "[20,40]"
```

Vamos a ver en detalle cómo hacer esto. Pero sería muy bueno que antes de ver la solución te tomes unos minutos para pensar cómo lo harías tú.

Una vez fijado el directorio de trabajo, leemos el fichero `Tut12-CrateresLunaEnBruto-01.csv` en un `data.frame` llamado `crateresBruto`. Hemos renombrado las variables columna, porque no nos gustaban los nombres que R asigna automáticamente a las últimas cuatro columnas (otra posibilidad es usar la opción `check.names` de la función `read.tables`):

```
crateresBruto = read.table(file="../datos/Tut12-CrateresLunaEnBruto-01.csv",header=TRUE,sep=",")
colnames(crateresBruto) = c("S", "N", "[20,40]", "(40,60]", "(60,80]", "[80, )")
```

Ahora, fusionamos las dos primeras columnas en el vector `hemisphere` así:

```
hemisphere = ifelse(crateresBruto[ ,1], yes = "SUR", no = "NORTE")
head(hemisphere)

## [1] "SUR"   "SUR"   "NORTE" "NORTE" "NORTE" "SUR"
```

Algunos comentarios sobre este código:

- Nos hemos aprovechado del hecho de que la información de las dos primeras columnas es redundante (si una es 1, la otra es 0 y viceversa) para trabajar sólo con la primera columna.
- Hemos usado la función `ifelse` de R, que evalúa una condición lógica y devuelve el valor del argumento `yes` cuando esa condición es cierta (en este ejemplo, devuelve la cadena de texto `"SUR"`), pero si es falsa devuelve el valor del argumento `no` (en este ejemplo, devuelve la cadena de texto `"NORTE"`). Además, hemos aprovechado el hecho de que R interpreta el número 1 como `TRUE`, y el número 0 como `FALSE`, directamente. Si quieres una versión más clara de lo que hemos hecho (sin esas conversiones implícitas de números en valores booleanos `TRUE/FALSE`), puedes emplear esta línea de código alternativa:

  ```
  hemisphere = ifelse( crateresBruto[,1] == 1, yes = "SUR", no = "NORTE")
  ```

  en la que explícitamente preguntamos si el valor de la primera columna es 1.

Puedes comprobar que la tabla de frecuencias es la que hemos mostrado antes:

```
table(hemisphere)

## hemisphere
## NORTE   SUR
##  2402  2783
```

¿Te has fijado en el orden en que aparecen las variables en esta tabla? ¿De qué tipo es la variable `hemisphere`?

Ejercicio 4. *Responde a esa pregunta. Queremos que la variable* `hemisphere` *sea un factor. Así que si no lo es, conviértela en un factor. Soluciones en la página 24.* □

La idea que vamos a usar para fusionar las cuatro últimas columnas en un vector `craterSize` es similar. Pero ahora, puesto que tenemos que inspeccionar las cuatro últimas columnas (de la tercera a la sexta) ya no hay la misma redundancia en los datos de la que nos podamos aprovechar. La primera idea que se te puede ocurrir es usar un bucle `for` para recorrer las filas de la matriz y, para cada una ellas, usar la función `which` para localizar cuál es igual a 1. Enseguida verás por qué le hemos puesto el nombre `primerIntento` al resultado:

```
primerIntento = c()
for(i in 1:nrow(crateresBruto)){
  datosCrater = crateresBruto[i, 3:6]
  craterSizeTipo = which(datosCrater == 1)
  primerIntento = c(primerIntento, craterSizeTipo)
}
```

Queremos aprovechar este ejemplo para inivitar al lector a reflexionar sobre la eficiencia del código en R. Hasta este punto, en los distintos tutoriales nos hemos limitado a resolver los problemas sin preocuparnos demasiado de esto. Eso es justificable cuando está aprendiendo y mientras los problemas son de un tamaño modesto, como los que hemos ido encontrando en este curso. Pero al enfrentarnos a problemas del mundo real, se hace cada vez más necesario preocuparnos de que nuestro código sea *eficiente*. En este sentido, en el anterior fragmento de código hemos hecho las cosas casi de la peor manera posible. En R es **esencial usar operaciones vectorializadas y evitar los bucles** (`for, while, repeat`) siempre que sea posible.

Para ver un ejemplo inicial sencillo de esto vamos a usar una herramienta que R pone a nuestra disposición para medir el tiempo de ejecución de un fragmento de código: la función `system.time`. Esta función se puede usar para obtener una medida objetiva de ese tiempo de ejecución. Para usarla, basta con incluir como argumento el fragmento de dódigo cuya eficiencia queremos analizar. Por ejemplo, vamos a generar un vector de 10000000 números aleatorios y luego vamos a sumarle 1 a cada uno de esos números. Pero vamos a hacerlo de dos maneras distintas. En la primera versión usaremos un bucle `for` para acceder uno a uno a los elementos del vector. En la segunda, la versión vectorializada, simplemente usamos la aritmética vectorial de R.

```
n = 10000000
v = rnorm(n)
system.time(
  for (i in 1:n){
    v[i] = v[i] + 1
  }
)

##    user  system elapsed
##  12.584   2.522  14.967

system.time(v <- v +1)

##    user  system elapsed
##   0.018   0.007   0.025
```

El tiempo que aparece en la respuesta se mide en segundos. Como puedes ver, la segunda versión es varios órdenes de magnitud más rápida que la primera.

Volviendo a la fusión de columnas en el fichero ¿ccómo podemos hacerlo de manera eficiente? Usando una operación vectorializada, en este caso mediante la familia de funciones `apply` (que incluye `apply`, pero también `mapply, sapply, lapply,...`). Para hacer eso defininos una función, a la que hemos llamado `claseCrater`, que codifica la operación que queremos hacer en cada fila de la tabla de datos:

```
claseCrater = function(datosCrater){
  craterSize = which(datosCrater == 1)
}
```

Una vez hecho esto, usamos `apply` para aplicar esta función al fragmento de `crateresBruto` que contiene la información sobre la clase del tamaño del crater. El argumento `MARGIN` sirve para indicarle a R que queremos trabajar *por filas*:

```
craterSize = apply(crateresBruto[, 3:6], MARGIN = 1, FUN = claseCrater)
```

El resultado es un vector con el mismo número de elementos que filas hay en la tabla y que contiene, en cada posición, el resultado de aplicar `claseCrater` a la correspondiente fila de la tabla. Podemos empezar por comprobar que el resultado de nuestro primer intento coincide con el de `apply`:

```
table(craterSize - primerIntento)

##
##    0
## 5185
```

Así que tenemos dos formas de resolver el mismo problema. ¿Cuál es mejor? Hay varios sentidos en los que podemos medir esto: la rapidez es uno de ellos, desde luego. Pero también podemos preocuparnos por la sencillez del código. En ese sentido, la solución con `apply` es más abstracta, pero precisamente por eso resulta más simple (cabe en dos líneas de código). Al principio te costará valorarlo, pero a medida que ganes experiencia programando con R empezarás a apreciar la ventaja que supone ese nivel de abstracción. Y, en cualquier caso, volviendo al asunto de la velocidad, dependiendo de la potencia del ordenador que estés usando es posible que hayas podido percibir directamente la diferencia entre ambos métodos. Para obtener una medida más objetiva podemos usar una herramienta que R pone a nuestra disposición para medir el tiempo de ejecución de un fragmento de código: la función . Esta función se puede usar para incluir como argumento el fragmento de dódigo cuya eficiencia queremos analizar. Por ejemplo, para nuestro primer intento es:

```
set.seed(2015)
system.time({
  primerIntento = c()
  for(i in 1:nrow(crateresBruto)){
    datosCrater = crateresBruto[i, 3:6]
    craterSizeTipo = which(datosCrater == 1)
    primerIntento = c(primerIntento, craterSizeTipo)
  }
})

##    user  system elapsed
##   0.966   0.098   1.070
```

El dato más útil aquí es `elpased`, el tiempo en segundos que hemos empleado (hay muchos detalles técnicos en juego en la interpreteación de estos valores, pero para un caso sencillo basta con comparar `elapsed`)

Mientras que para el segundo es:

```
system.time({
  claseCrater = function(datosCrater){
    craterSize = which(datosCrater == 1)
  }
craterSize = apply(crateresBruto[, 3:6], MARGIN = 1, FUN = claseCrater)
})

##    user  system elapsed
##   0.024   0.003   0.026
```

La diferencia no es tan espectacular como antes, pero aún así es muy grande, de órdenes de magnitud. Confiamos en que este simple ejemplo haya estimulado tu curiosidad y te sirva como motivación para tratar de profundizar en el tema de la eficiencia en programación.

Volviendo a nuestro problema, la tabla de frecuencias de `craterSize` es la esperada:

```
table(craterSize)

## craterSize
```

```
##    1    2    3    4
## 3003 1108  482  592
```

Un detalle: tal como lo hemos construido, el vector `craterSize` no es un factor.

```
class(craterSize)

## [1] "integer"
```

Pero podemos convertirlo fácilmente en un factor:

```
craterSize = factor(craterSize,
                labels = c("[20,40]", "(40,60]", "(60,80]", "[80, )"))
```

Ahora que ya tenemos los dos vectores, con los niveles de ambos factores, obtener la tabla de contingencia (son sus valores marginales), y el contraste χ^2 de homogeneidad es muy fácil. Basta con usar:

```
addmargins(table(hemisphere, craterSize))

##           craterSize
## hemisphere [20,40] (40,60] (60,80] [80, )  Sum
##      NORTE    1388     523     227    264 2402
##      SUR      1615     585     255    328 2783
##      Sum      3003    1108     482    592 5185

chisq.test(hemisphere, craterSize)

##
##  Pearson's Chi-squared test
##
## data:  hemisphere and craterSize
## X-squared = 1.18, df = 3, p-value = 0.76
```

Ejercicio 5. *¿Cuál es la hipótesis nula, y cuál la conclusión del contraste que acabamos de pedirle a R que haga?* □

Ahora podemos ver más claramente lo que significa la idea de *datos limpios* para un contraste χ^2 como este. Si usamos los vectores `hemisphere` y `craterSize` para crear un `data.frame` llamado `craterDatosLimpios`, de esta manera:

```
craterDatosLimpios = data.frame(hemisphere, craterSize)
```

entonces las primeras filas de este `data.frame` son:

```
head(craterDatosLimpios,10)

##    hemisphere craterSize
## 1         SUR    (60,80]
## 2         SUR    [20,40]
## 3       NORTE    (60,80]
## 4       NORTE    [20,40]
## 5       NORTE    (40,60]
## 6         SUR    (40,60]
## 7         SUR     [80, )
## 8         SUR    [20,40]
## 9         SUR    (40,60]
## 10        SUR    [20,40]
```

Y confirman que estamos ante una estructura de datos limpia, en el sentido que venimos utilizando:

- Una fila para cada observación individual.
- Una columna para cada variable.

Si ahora queremos guardar estos datos limpios en un fichero `csv`, basta con usar: y obtendremos el fichero que queríamos.

```
write.table(craterDatosLimpios, file="../datos/Tut12-CrateresLunaEnBruto.csv",
    sep=",", row.names=FALSE, col.names=TRUE)
```

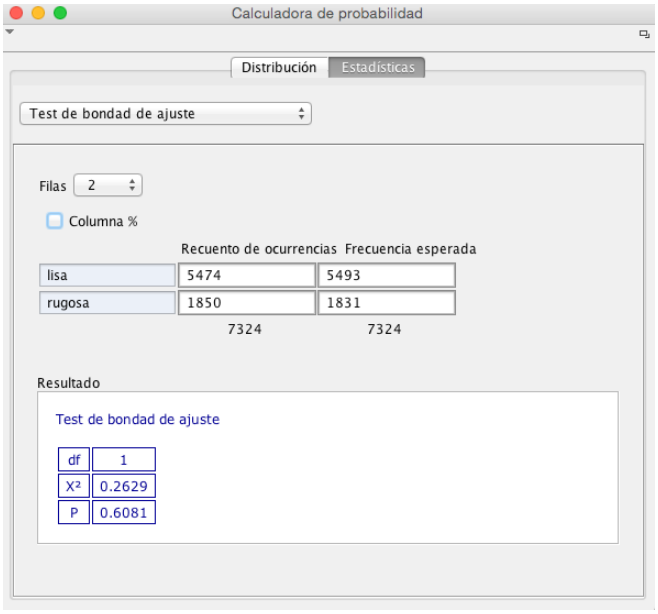
3. Contrastes χ^2 en otros programas.

3.1. GeoGebra.

En GeoGebra es muy fácil hacer contrastes χ^2 de independencia usando la *Calculadora de Probabilidades* (pestaña *Estadísticas*). Por ejemplo, en la siguiente figura tienes el resultado que se obtiene para los datos del Ejemplo del *Barómetro del CIS*.

Ejercicio 6. *Usa GeoGebra para comprobar los resultados del Ejemplo 12.1.5 del libro (pág. 473), el de las poblaciones de Avutardas.* □

Y también es posible obtener un contraste de homogeneidad, como hemos hecho en la siguiente figura para el Ejemplo 12.2.4 del libro (pág. 483), el de Mendel y sus guisantes.



Si en alguna ocasión necesitas o prefieres usar la *Línea de Entrada* o la vista *CAS*, GeoGebra pone a nuestra disposición la función `TestChiCuadrado`, con la que podemos hacer todos estos contrastes. No dejes de consultar la ayuda de esa función antes de utilizarla.

3.2. Calc.

Queremos comentar, muy brevemente, las posibilidades que ofrece la hoja de cálculo Calc para este tipo de contrastes. Aunque a estas alturas del curso esperamos haber persuadido al lector de las ventajas de un software especializado (como R), en alguna ocasión puede ser necesario utilizar esta herramienta básica.

Calc nos ofrece la función `PRUEBA.CHI` (`CHITEST`, en la versión en inglés de Calc) para este tipo de contrastes. En la Figura 3 puedes ver como la usamos para el Ejemplo 12.1.1 del libro, el del *Barómetro* del CIS.

La función `PRUEBA.CHI` usa dos tablas de valores, almacenadas en sendos rangos de celdas, para los valores observados y esperados. ¿Los grados de libertad? Se determinan a partir de las dimensiones (número de filas y columnas) de esas tablas (que desde luego, deben coincidir). El resultado es el p-valor del contraste que, como puedes ver, coincide con el valor que hemos obtenido en R. Si se desea calcular el valor del estadístico del contraste, tenemos que hacerlo manualmente. Por ejemplo, en la celda `A11` de la Figura 3 nosotros hemos incluido el código:

```
=(A3-D3)^2/D3
```

y hemos hecho lo propio en el rango `A11:B12`. El estadístico se calcula después sumando los valores de ese rango de residuos estudentizados. El fichero adjunto

Tut12-BarometroCIS.ods

contiene esa hoja de cálculo, que con algunas modificaciones puede adaptarse a otros ejemplos.

¿Y si lo que queremos es hacer un contraste χ^2 de homogeneidad? Usamos la misma función `PRUEBA.CHI`. Basta, en ese caso, con que los valores observados y esperados ocupen rangos unidimensionales (cada uno de ellos en una fila o una columna). Por ejemplo, en la Figura 4 puedes ver esa función aplicada a los datos del Ejemplo 12.2.1 del libro, el del dado cargado.

De nuevo, si se desea el valor del estadístico, es necesario calcularlo *a mano*. El fichero de este ejemplo es:

Tut12-DadoCargado.ods

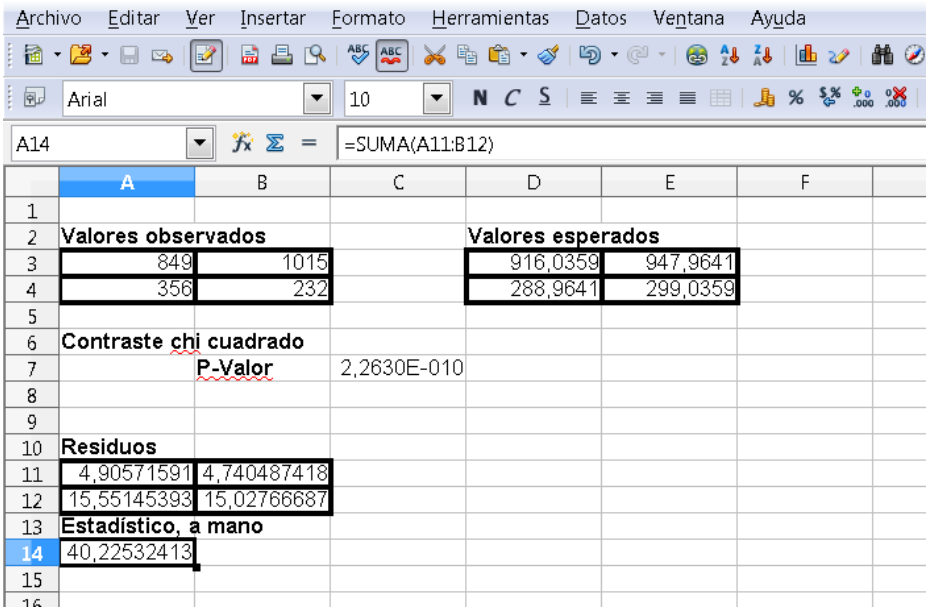


Figura 3: La función `PRUEBA.CHI` de Calc, para el contraste χ^2 de independencia con los datos del Barómetro del CIS

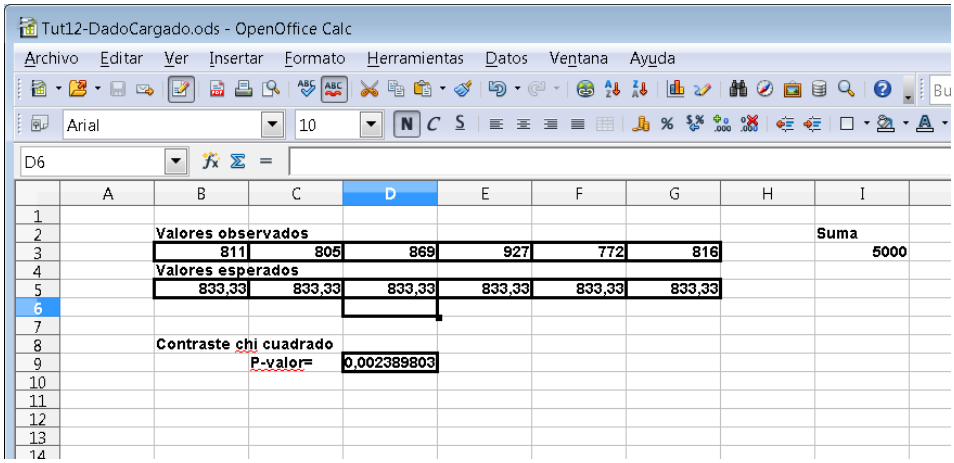


Figura 4: La función `PRUEBA.CHI` de Calc, para el contraste χ^2 de homogeneidad con los datos del Ejemplo 12.2.1 del libro.

Ejercicio:
Usa Calc para obtener el estadístico de este último ejemplo. Y calcula, también con Calc, los p-valores de los Ejemplos 12.1.5 (avutardas) y 12.2.4 (Mendel) del libro. □

4. El contraste exacto de Fisher. Distribución hipergeométrica.

Vamos a ver como utilizar R para obtener los resultados del Ejemplo 12.3.1 del libro (pág. 485) y de sus continuaciones en los ejemplos del 12.3.2 hasta el 12.3.7 (pág. 498), en el que finalmente hemos obtenido el p-valor del contraste. A estas alturas, el lector seguramente espera que R tenga funciones casi para cualquier cosa. Así que empecemos esta vez por lo más sencillo. Para obtener ese p-valor que aparece en el Ejemplo 12.3.7 basta con aplicar la función `fisher.test` a los datos de la tabla de contingencia observada:

```
(fisherTest = fisher.test(matrix(c(9, 6, 3, 12), nrow = 2), alternative = "greater"))
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  matrix(c(9, 6, 3, 12), nrow = 2)
## p-value = 0.03
## alternative hypothesis: true odds ratio is greater than 1
## 95 percent confidence interval:
##  1.1833    Inf
## sample estimates:
## odds ratio
##     5.6137
```

Como ves, el p-valor coincide con el que hemos obtenido en el ejemplo del libro. Ahora queremos reconstruir paso a paso el cálculo manual de ese p-valor. Pero antes, dado que la vamos a necesitar, veamos brevemente como se trabaja en R con la distribución hipergeométrica.

La distribución hipergeométrica en R.

El lector no se sorprenderá si le decimos que R dispone del cuarteto habitual de funciones `d, p, q, r`, en este caso identificadas por el sufijo `hyper`. Recuerda que pensamos en una caja con N bolas, de las cuales B son blancas. Extraemos una muestra de m bolas de la caja, **sin reemplazamiento**, y nos preguntamos por la probabilidad de que k de las bolas extraídas sean blancas.

Esta es la notación que hemos usado en el libro, porque creemos que aporta claridad. Desafortunadamente, en R las funciones relacionadas con la hipergeométrica usan una notación distinta y, hasta cierto punto, conflictiva con la que usamos nosotros. Por ejemplo, en la función

```
dhyper(x, m, n, k)
```

la ayuda de R nos indica que:

- `m` representa el número de bolas blancas en la urna. Lo hemos llamado B en el libro.
- `n` representa el número de bolas negras en la urna. En el libro no tiene un nombre asignado, es $N - B$.
- `k` representa el número de bolas que se extraen de la caja. Aquí es donde hay más riesgo de confusión, porque nosotros lo hemos llamado m.
- `x` es el número de bolas (blancas entre las extraídas) del que queremos calcular la probabilidad. Y que, para aumentar la confusión, nosotros llamamos k.

Para no perdernos en este embrollo de símbolos recomendamos encarecidamente el uso del tabulador en RStudio y mantener a la vista la Tabla 12.17 del libro (pág. 497). De esa forma, cuando uses R, podrás asegurarte de que asignas a cada parámetro el valor que realmente deseas.

Si, por ejemplo, trabajamos con una distribución hipergeométrica $Hyp(N = 200, B = 130, m = 50)$, entonces la probabilidad de extraer exactamente $k = 17$ bolas blancas es

```
dhyper(x = 17, m = 130, n = 200 - 130, k = 50)
```

```
## [1] 1.7995e-07
```

mientras que la probabilidad de obtener 17 o menos es:

```
phyper(q = 17, m = 130, n = 200 - 130, k = 50)
```

```
## [1] 2.1392e-07
```

El cuantil 0.75 de esa distribución hipergeométrica se obtiene con:

```r
qhyper(p = 0.75, m = 130, n = 200 - 130, k = 50)

## [1] 34
```

Ejercicio 7. *Interpreta este resultado en términos de probabilidad.* □

Finalmente, si lo que queremos es obtener valores aleatorios de esa distribución, por ejemplo para simular varias extracciones de muestras del mismo tamaño de esa caja, usaremos `rhyper`:

```r
rhyper(20, m = 130, n = 200 - 130, k = 50)

##  [1] 34 36 32 31 30 35 29 30 35 38 32 30 33 31 36 34 32 31 30 33
```

El resultado indica el número de bolas blancas que contiene cada una de las 20 muestras de 50 bolas que hemos extraído.

El contraste de Fisher paso a paso.

Volvamos al Ejemplo del libro. Empezamos con la tabla de valores observados, que aparece en el Ejemplo 12.3.3 (pág. 489), a la que le añadimos las decoraciones habituales:

```r
(tablaObservada = matrix( c(9, 3, 6, 12),  nrow= 2, byrow = TRUE))

##      [,1] [,2]
## [1,]    9    3
## [2,]    6   12

colnames(tablaObservada) = c("Expuestos", "NoExpuestos")
rownames(tablaObservada) = c("Enfermos", "Sanos" )
addmargins(tablaObservada)

##          Expuestos NoExpuestos Sum
## Enfermos         9           3  12
## Sanos            6          12  18
## Sum             15          15  30
```

Llamemos a esta tabla `tabla0`.

```r
tabla0 = tablaObservada
```

Las tres tablas muestrales que tenemos que considerar son las que aparecen a continuación:

```r
(tabla1 = matrix( c(10, 2, 5, 13),  nrow= 2, byrow = TRUE))

##      [,1] [,2]
## [1,]   10    2
## [2,]    5   13

colnames(tabla1) = c("Expuestos", "NoExpuestos")
rownames(tabla1) = c("Enfermos", "Sanos" )
addmargins(tabla1)

##          Expuestos NoExpuestos Sum
## Enfermos        10           2  12
## Sanos            5          13  18
## Sum             15          15  30
```

```r
(tabla2 = matrix( c(11, 1, 4, 14),  nrow= 2, byrow = TRUE))
```

```
##      [,1] [,2]
## [1,]   11    1
## [2,]    4   14
```

```r
colnames(tabla2) = c("Expuestos", "NoExpuestos")
rownames(tabla2) = c("Enfermos", "Sanos" )
addmargins(tabla2)
```

```
##          Expuestos NoExpuestos Sum
## Enfermos        11           1  12
## Sanos            4          14  18
## Sum             15          15  30
```

```r
(tabla3 = matrix( c(12, 0, 3, 15),  nrow= 2, byrow = TRUE))
```

```
##      [,1] [,2]
## [1,]   12    0
## [2,]    3   15
```

```r
colnames(tabla3) = c("Expuestos", "NoExpuestos")
rownames(tabla3) = c("Enfermos", "Sanos" )
addmargins(tabla3)
```

```
##          Expuestos NoExpuestos Sum
## Enfermos        12           0  12
## Sanos            3          15  18
## Sum             15          15  30
```

A cada una de estas tablas le corresponde una probabilidad que podemos calcular usando una distribución hipergeométrica. Los parámetros de la distribución hipergeométrica y la correspondiente probabilidad se obtienen así a partir de la tabla observada:

```r
(N = sum(tabla0))
```

```
## [1] 30
```

```r
(B = sum(tabla0[,1]))
```

```
## [1] 15
```

```r
(m = sum(tabla0[1, ]))
```

```
## [1] 12
```

```r
(pTabla0 = dhyper(x=tabla0[1,1], m = B, n = N - B, k = m))
```

```
## [1] 0.026329
```

Y para las otras tres tablas, de forma similar

```r
(pTabla1 = dhyper(x=tabla1[1,1], m = B, n = N - B, k = m))
```

```
## [1] 0.0036455
```

```
####################

(pTabla2 = dhyper(x=tabla2[1,1], m = B, n = N - B, k = m))

## [1] 0.00023672

####################

(pTabla3 = dhyper(x=tabla3[1,1], m = B, n = N - B, k = m))

## [1] 0.0000052605
```

Y con eso el p-valor del contraste de Fisher es:

```
pTabla0 + pTabla1 + pTabla2 + pTabla3

## [1] 0.030216
```

que, como puedes comprobar, coincide con el que nos ha proporcionado la función `fisher.test`.

GeoGebra.

Apenas una línea para comentar que en GeoGebra existen las funciones `Hipergeométrica` e `HipergeométricaInversa` que juegan un papel similar a `phyper` y `qhyper`. No existe, que nosotros conozcamos, una función que permita calcular directamente un contraste de Fisher.

5. Ejercicios adicionales y soluciones.

Ejercicios adicionales.

Ejercicio 8. *La competición entre tratamientos (de segunda generación) para el alicaimiento en frailecillos se ha reducido a dos finalistas: Plumacetamol y Picozepán. Para decidir cuál de estos dos es mejor, el laboratorio ACME-Farma ha realizado un experimento, en el que han intervenido 400 frailecillos alicaídos, que se han repartido al azar entre dos grupos tratados de forma independiente. Al primer grupo, formado por 175 frailecillos, se le ha suministrado Plumacetamol. Al segundo grupo, formado por el resto de los frailecillos, los hemos tratado con Picozepán. Sabemos que se han curado en total 250 frailecillos, de los que 130 habían sido tratados con Plumacetamol.*

1. *Completa la tabla de doble entrada que aparece debajo, y úsala para contestar a la pregunta: sabiendo que un frailecillo se ha curado, ¿cuáles son las probabilidades de que haya sido tratado con cada uno de los dos tratamientos?*

	Curados	*No Curados*	*Total*
Plumacetamol	*??*	*??*	*??*
Picozepán	*??*	*??*	*??*
Total	*??*	*??*	*??*

2. *Haz un contraste χ^2 (puedes usar 95 % como nivel de significación) para decidir si los dos tratamientos son igual de eficaces.*

3. *Haz de nuevo el contraste, pero en este caso, utilizando los métodos del Capítulo 9 del libro. ¿Hay alguna relación entre los valores de los estadísticos que has usado en esos dos contrastes?*

4. *Además, han surgido dudas sobre el procedimiento por el que los frailecillos se han asignado a los dos grupos. Suponiendo que cada frailecillo tenía la misma probabilidad de ser tratado con cada medicamento, ¿dirías que el número de frailecillos asignado a cada tratamiento es fruto del azar?*

□

Soluciones de algunos ejercicios.

• **Ejercicio 4 (pág. 13).**

Utiliza la función `factor` así:

```
class(hemisphere)
hemisphere = factor(hemisphere)
class(hemisphere)
```

Fin del Tutorial12. ¡Gracias por la atención!