Ejercicio 4

Cedric Prieels

26/10/2016

LINEA 140

Resumen

Este ejercicio consiste en estudiar un fichero de datos que contiene el número de noches de observación astrónomica perdidas por culpa del tiempo en diferentes meses de algunos años, con diferentes métodos estadísticos vistos en clase. El objetivo principal es ajustar esta serie de datos a diferentes funciones : a una recta de pendiente 0, a una recta más general y a un modelo periodico, para ver que tipo de modelo queda el mejor en cada caso.

Metodología

En este ejercicio, estudiamos un fichero de datos que contiene una tabla con el número de noches de observación astronómica perdidas por culpa del tiempo, en función del mes y del año, entre 1894 y 2005. El ejercicio se divida en dos partes distintas : en la primera parte, estudiamos la fracción de noches perdidas por año (hacemos una media por linea) y en la segunda parte, miramos la fracción de noches perdidas por mes (media por columna).

Primera parte

En la primera parte, creamos primero una nueva tabla que tiene la fracción de noches perdidas por año y su error estimado (error debido al cálculo de la media), quitando los *ND* que aparecen en la tabla original cuando no hay datos disponibles. Una vez esta tabla obtenida, pintamos sus datos y sus errores en un plot de tipo *errbar*.

Tenemos que intentar modelizar los datos que tenemos, usando dos modelos distintos. Primero, intentamos ajustar une recta constante (de pendiente 0) a los datos usando diferentes métodos, para ver el valor del chi cuadrado y de bondad que nos devuelve este ajuste. Después, volvemos a repetir lo mismo pero usando un modelo teórico un poco diferente : intentamos ajustar a los datos una recta más general que tenga una pendiente distinta de 0. En este caso también volvemos a calcular el valor del chi2 (y su intervalo de confianza del 90%) para ver si este modelo queda mejor que el primero, o no. Esta comparación entre ajustes se acaba con un F test, para encontrar el modelo que se ajusta lo mejor a nuestros datos (por supuesto, esperamos que el segundo modelo quede mejor porque tiene un parámetro libre suplementario, la pendiente de la recta).

Segunda parte

En la segunda parte, repetimos el mismo proceso pero usando datos un poco distintos : en lugar de usar la fracción de noches perdidas por año, usamos en este caso la fracción de noches perdidas por mes. Creamos entonces esta tabla primero, haciendo una media por columna de la tabla original de datos.

Después, intentamos ajustar un modelo periodico que tenga tres parámetros a estos datos (los datos ya no siguen una distribución lineal, porque el tiempo depende del mes considerado, y esperamos por ejemplo tener un número de noches perdidas mayor en invierno que en verano), minimizando el valor del chi cuadrado y

calculando su intervalo de confianza del 90% también. En este caso, este intervalo del chi2 no es un solo valor pero un contorno en el espacio bidimensional, porque tenemos tres parámetros a nuestro modelo : dos importantes y la fase, que fijamos.

Resultados

Primera parte

Primero, abrimos el fichero que contiene los datos, los números de noche de observación astronomica perdidas por culpa del tiempo, por cada mes de algunos años. Representamos los datos así obtenidos en una tabla, para ver que pinta tienen.

```
rm(list=ls())

setwd('/Users/ced2718/Documents/Universite/Estadistica/Ejercicio_4/')
data <- read.table("Ejercicio4_Datos.dat", header=T, na.strings = "ND")
data
```

```
##       Ene  Feb  Mar  Abr  May Jun Jul Ago  Sep  Oct  Nov  Dic
## 1984   NA   NA   NA   NA  0.5 2.5 3.0 0.5  5.0  2.5  9.0  7.5
## 1985 15.5 14.5  6.5  7.0  7.0 2.0 0.0 3.0  5.5  6.5 13.5  2.5
## 1986 11.0 13.0  7.5  6.5  1.0 1.0 0.0 0.5  1.5  5.5  4.5  3.5
## 1987 11.0  2.0 11.0 13.0  8.0 2.0 0.0 1.0 11.5 12.0  4.5  4.0
## 1988  8.5  6.5  1.0  5.0  5.5 2.0 1.5 1.5  4.5 11.0  5.0  9.5
## 1989  6.0 12.5  8.0  5.0  5.0 0.5 2.0 4.0  6.5 17.0 18.0 17.5
## 1990  8.5  2.0 14.0 15.5  6.5 2.5 3.0 2.0 12.0  5.5  9.5 15.5
## 1991  2.5  6.5 13.0  6.0  3.0 1.0 0.0 0.5  6.0 11.5 13.5 15.0
## 1992 10.5 14.0  8.5  5.0  3.0 3.0 0.0 2.0  3.5 14.0  6.5 15.0
## 1993  8.0 10.0 11.5  3.5 11.5 0.0 0.0 1.5  7.5  9.0 14.5  7.0
## 1994 14.5  4.5  5.5  6.0  4.0 1.0 1.0 1.0  4.0 12.5  7.0  6.0
## 1995  3.5  4.5 15.5  1.0  3.5 1.5 1.0 5.0  8.0  7.5 16.0 13.0
## 1996 18.5 13.5 16.0 10.0 14.5 0.0 1.0 3.0  8.0  6.0 11.0 15.0
## 1997 20.5  8.5 10.5 10.5  8.0 2.5 1.0 2.5  8.0 10.0  9.0  1.5
## 1998   NA  6.0 10.0  1.0  5.0  NA  NA  NA  1.5  2.0  3.5  4.0
## 1999 24.0  1.5  6.5  3.5   NA 0.5 0.0 1.5  2.0  6.1 20.0  9.7
## 2000 20.7  4.6  3.2  8.5  7.4  NA  NA 1.1  1.0  4.5  2.0  7.5
## 2001  0.0  1.5  7.0  4.3  0.8  NA  NA 0.5 11.3 16.0 16.3 15.0
## 2002 15.5 12.0 15.0 16.0  6.0 1.5 1.0 3.5 10.5  5.0 11.0 13.5
## 2003 11.6  8.8  9.8  9.3  2.3 1.5 2.0 3.8  2.4 16.5  6.0  5.5
## 2004  6.0 17.5 10.0 17.5 10.0 4.0 6.0 3.0 11.0 15.0 16.5 16.5
## 2005 12.5 28.0 11.5  6.5  8.0 3.0 4.5 1.0   NA   NA   NA   NA
```

Con esta tabla, podemos observar que tenemos datos entre Mayo 1984 y Agosto 2005. Vemos que el número de noches perdidas en cada mes se representa con un número decimal, redondeo a la media unidad o a la unidad. También se puede ver que algunos meses no tienen medida asociada (esto está representado por el texto *ND*, que ya hemos convertido a un string *NA* que R puede entender fácilmente).

El primer objetivo del ejercicio consiste en manipular un poco esta tabla para llegar a tener una tabla con tres columnas : el año, el percentage de noches perdidas y el error asociado a la medida (en este caso, suponemos que este error es el error debido al cálculo de la media). Lo díficil a la hora de calcular este percentage es tener en cuenta el número de días que cada mes tiene (y por supuesto, tener en cuenta los años bisextiles que tienen un día mas en febrero). El paquete *Hmisc* nos ayuda en este caso, porque tiene una función para calcular el número de días que tiene cada mes de cada año implementada.

```
require(Hmisc)
```

```
## Loading required package: Hmisc

## Loading required package: lattice

## Loading required package: survival

## Loading required package: Formula

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 3.3.2

##
## Attaching package: 'Hmisc'

## The following objects are masked from 'package:base':
##
##     format.pval, round.POSIXt, trunc.POSIXt, units
```

```
getNumberDaysPerMonth <- function(month, year) {
  return(monthDays(as.Date(paste('01', month, year, sep="-"),format='%d-%m-%Y')))
}
```

Podemos hacer un chequeo rapido de esta función para verificar que funcione correctamente, incluso en el caso de los años bisextos por ejemplo.

```
getNumberDaysPerMonth(02, 2015)
```

```
## [1] 28
```

```
getNumberDaysPerMonth(02, 2016)
```

```
## [1] 29
```

```
getNumberDaysPerMonth(11, 2016)
```

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

Primero, vamos a crear un nuevo dataframe llamado *fractionLost* que es del mismo formato que el dataframe *data* que contiene nuestros datos, excepto que queremos que cada celda contenga ahora la fracción de noches perdidas por mes, y no el número de noches perdidas.

```
years <- as.numeric(row.names(data))
fractionLost <- data.frame(row.names = row.names(data))
for(i in 1:length(years)) {
  for(j in 1:12) {
    fractionLost[i,j] <- data[i,j] / getNumberDaysPerMonth(j, years[i])
  }
}
colnames(fractionLost) <- colnames(data)
fractionLost
```

```
##             Ene        Feb        Mar        Abr        May        Jun
## 1984         NA         NA         NA         NA 0.01612903 0.08333333
## 1985 0.50000000 0.51785714 0.20967742 0.23333333 0.22580645 0.06666667
## 1986 0.35483871 0.46428571 0.24193548 0.21666667 0.03225806 0.03333333
## 1987 0.35483871 0.07142857 0.35483871 0.43333333 0.25806452 0.06666667
## 1988 0.27419355 0.22413793 0.03225806 0.16666667 0.17741935 0.06666667
## 1989 0.19354839 0.44642857 0.25806452 0.16666667 0.16129032 0.01666667
## 1990 0.27419355 0.07142857 0.45161290 0.51666667 0.20967742 0.08333333
## 1991 0.08064516 0.23214286 0.41935484 0.20000000 0.09677419 0.03333333
## 1992 0.33870968 0.48275862 0.27419355 0.16666667 0.09677419 0.10000000
## 1993 0.25806452 0.35714286 0.37096774 0.11666667 0.37096774 0.00000000
## 1994 0.46774194 0.16071429 0.17741935 0.20000000 0.12903226 0.03333333
## 1995 0.11290323 0.16071429 0.50000000 0.03333333 0.11290323 0.05000000
## 1996 0.59677419 0.46551724 0.51612903 0.33333333 0.46774194 0.00000000
## 1997 0.66129032 0.30357143 0.33870968 0.35000000 0.25806452 0.08333333
## 1998         NA 0.21428571 0.32258065 0.03333333 0.16129032         NA
## 1999 0.77419355 0.05357143 0.20967742 0.11666667         NA 0.01666667
## 2000 0.66774194 0.15862069 0.10322581 0.28333333 0.23870968         NA
## 2001 0.00000000 0.05357143 0.22580645 0.14333333 0.02580645         NA
## 2002 0.50000000 0.42857143 0.48387097 0.53333333 0.19354839 0.05000000
## 2003 0.37419355 0.31428571 0.31612903 0.31000000 0.07419355 0.05000000
## 2004 0.19354839 0.60344828 0.32258065 0.58333333 0.32258065 0.13333333
## 2005 0.40322581 1.00000000 0.37096774 0.21666667 0.25806452 0.10000000
##             Jul        Ago        Sep        Oct        Nov        Dic
## 1984 0.09677419 0.01612903 0.16666667 0.08064516 0.30000000 0.24193548
## 1985 0.00000000 0.09677419 0.18333333 0.20967742 0.45000000 0.08064516
## 1986 0.00000000 0.01612903 0.05000000 0.17741935 0.15000000 0.11290323
## 1987 0.00000000 0.03225806 0.38333333 0.38709677 0.15000000 0.12903226
## 1988 0.04838710 0.04838710 0.15000000 0.35483871 0.16666667 0.30645161
## 1989 0.06451613 0.12903226 0.21666667 0.54838710 0.60000000 0.56451613
## 1990 0.09677419 0.06451613 0.40000000 0.17741935 0.31666667 0.50000000
## 1991 0.00000000 0.01612903 0.20000000 0.37096774 0.45000000 0.48387097
## 1992 0.00000000 0.06451613 0.11666667 0.45161290 0.21666667 0.48387097
## 1993 0.00000000 0.04838710 0.25000000 0.29032258 0.48333333 0.22580645
## 1994 0.03225806 0.03225806 0.13333333 0.40322581 0.23333333 0.19354839
## 1995 0.03225806 0.16129032 0.26666667 0.24193548 0.53333333 0.41935484
## 1996 0.03225806 0.09677419 0.26666667 0.19354839 0.36666667 0.48387097
## 1997 0.03225806 0.08064516 0.26666667 0.32258065 0.30000000 0.04838710
## 1998         NA         NA 0.05000000 0.06451613 0.11666667 0.12903226
## 1999 0.00000000 0.04838710 0.06666667 0.19677419 0.66666667 0.31290323
## 2000         NA 0.03548387 0.03333333 0.14516129 0.06666667 0.24193548
## 2001         NA 0.01612903 0.37666667 0.51612903 0.54333333 0.48387097
## 2002 0.03225806 0.11290323 0.35000000 0.16129032 0.36666667 0.43548387
## 2003 0.06451613 0.12258065 0.08000000 0.53225806 0.20000000 0.17741935
## 2004 0.19354839 0.09677419 0.36666667 0.48387097 0.55000000 0.53225806
## 2005 0.14516129 0.03225806         NA         NA         NA         NA
```

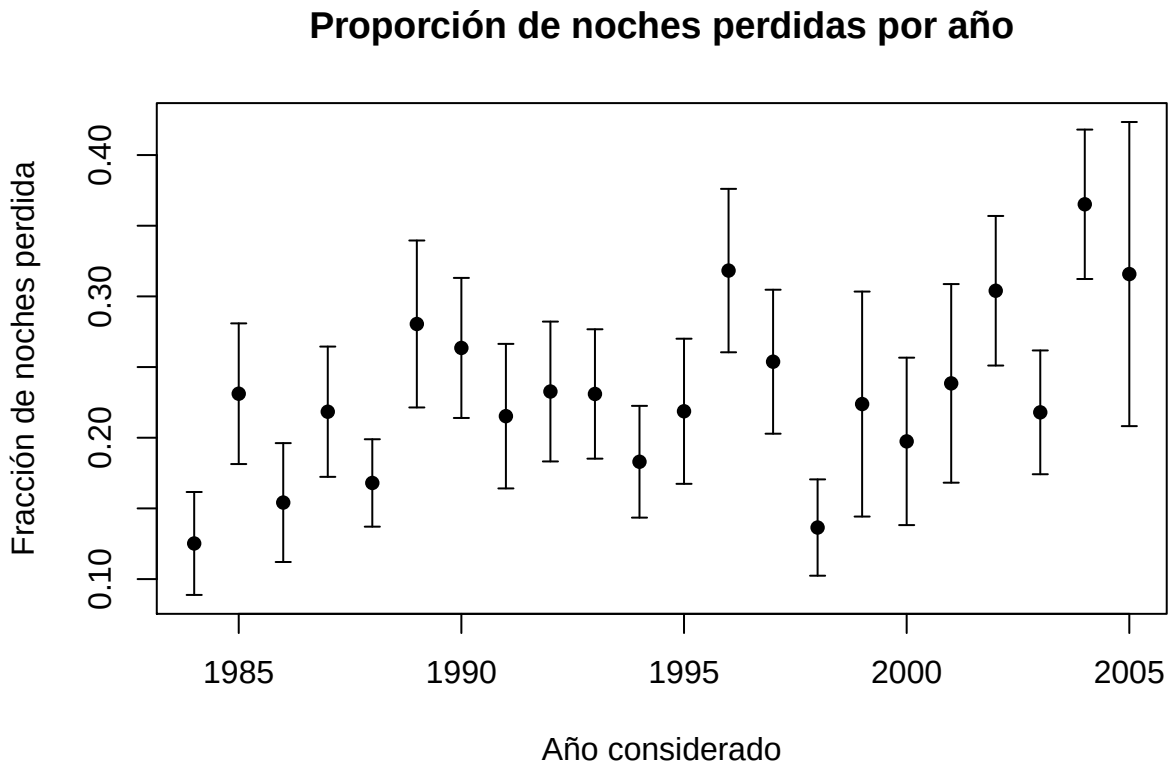
Ahora, llegar a la tabla que queremos obtener es muy fácil. Solo tenemos que calcular la media de la proporción de noches perdida de cada año, y de calcular el error asociado (el error de una media se calcula dividiendo la desviación estándar de los datos por la raíz del número de elementos que tenemos, que no sean del tipo *NA*).

```r
fractionLostByYear <- c()
estimatedError <- c()
fractionLostByYear <- rowMeans(fractionLost, na.rm = T)
for(i in 1:length(years)) {
  estimatedError[i] <- sd(fractionLost[i,1:12], na.rm = T)/sqrt(length(which(!is.na(data[i,]))))
}
newData <- data.frame(years, fractionLostByYear, estimatedError)
newData
```

```
##      years fractionLostByYear estimatedError
## 1984  1984          0.1252016     0.03641051
## 1985  1985          0.2311476     0.04975791
## 1986  1986          0.1541475     0.04205410
## 1987  1987          0.2184076     0.04608115
## 1988  1988          0.1680061     0.03092405
## 1989  1989          0.2804820     0.05906778
## 1990  1990          0.2635241     0.04954609
## 1991  1991          0.2152682     0.05113503
## 1992  1992          0.2327030     0.04947453
## 1993  1993          0.2309716     0.04577177
## 1994  1994          0.1830165     0.03952987
## 1995  1995          0.2187244     0.05133462
## 1996  1996          0.3182734     0.05780758
## 1997  1997          0.2537922     0.05092741
## 1998  1998          0.1364631     0.03408173
## 1999  1999          0.2238340     0.07959213
## 2000  2000          0.1974212     0.05921931
## 2001  2001          0.2384647     0.07026017
## 2002  2002          0.3039939     0.05291902
## 2003  2003          0.2179647     0.04378841
## 2004  2004          0.3651619     0.05286236
## 2005  2005          0.3157930     0.10757559
```

Podemos también pintar estos datos que acabamos de calcular para ver la pinta que tienen, usando el package HMSIC y su función errbar para poder pintar en el plot las barras de errores que hemos calculado también.

```r
errbar(years, fractionLostByYear,
       yplus = fractionLostByYear + estimatedError, ymin = fractionLostByYear - estimatedError,
       xlab = "Año considerado", ylab = "Fracción de noches perdida")
title(main = "Proporción de noches perdidas por año")
```



Ajuste a una constante

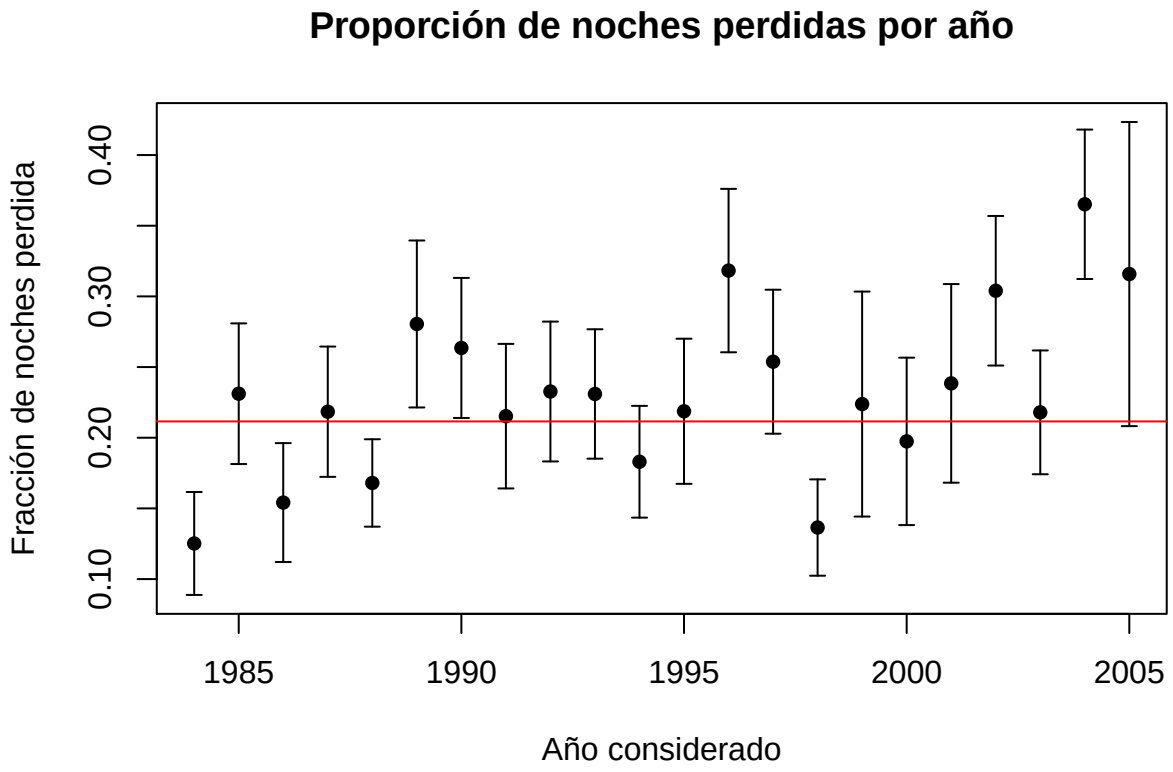
Ahora que hemos calculado la fracción de noches de observación perdidas por año, podemos intentar modelizar estos datos por una recta constante, que tenga un pendiente igual a 0. Esto se puede hacer de diferentes maneras. La primera manera consiste en usar el método de R *lm* pesado (porque tenemos errores, y los pesos valen la varianza de este error estimado en la tabla precediente).

```
firstModelCoef <- lm(fractionLostByYear ~ 1, weights = 1/estimatedError^2)
firstModelCoef$coefficients
```

```
## (Intercept)
##   0.2115525
```

En este caso, podemos ver que la recta que imita lo mejor nuestros datos es la recta que tiene un valor constante igual a 0.2115525. La podemos añadir al plot precediente para ver la pinta que tiene este ajuste.

```
errbar(years, fractionLostByYear,
       yplus = fractionLostByYear + estimatedError, ymin = fractionLostByYear - estimatedError,
       xlab = "Año considerado", ylab = "Fracción de noches perdida")
title(main = "Proporción de noches perdidas por año")
abline(h=firstModelCoef$coefficients, col="red")
```



Ya vemos que aproximar nuestros datos por una recta de pendiente nula permite dar una idea de la pinta que tienen los datos, pero nada más, porque no es muy preciso (vemos que algunos puntos se alejan bastante, hasta más o menos dos sigmas, de la recta constante roja). Podemos también verificar si la ecuación de la recta que acabamos de obtener es correcta o no, usando un otro método de cálculo (por el uso de *weighted.mean*) muy similar al método precediente, que tiene que devolver exactamente lo mismo.

```
weighted.mean(fractionLostByYear, 1/estimatedError^2)
```

```
## [1] 0.2115525
```

Ahora, podemos calcular el valor de chi cuadrado del ajuste que acabamos de calcular. El chi2 corresponde a los datos (fracción perdida de noches) menos el valor devuelto por el modelo (aquí, vale una constante) y dividido por el error de la tabla, todo al cuadrado. Calculamos este valor definiendo una nueva función, capaz de calcular chi2 para diferentes datos y modelos. También podemos calcular el intervalo de confianza del 90% de este valor, usando la expresión que vimos en clase (un cálculo directo se puede hacer en este caso, como solo tenemos un parámetro interesante).

```
getChi2 <- function(data, model, error) {
  punto <- ((data-model)/error)^2
  return(sum(punto))
}
```

```
chi2Model1 <- getChi2(fractionLostByYear, firstModelCoef$coefficients, estimatedError)
chi2Model1
```

```
## [1] 34.65221
```

```r
error90Chi2 <- sqrt(2.71/sum(1/estimatedError^2))
error90Chi2
```

```
## [1] 0.01668364
```

Ahora que tenemos estos valores, queremos calcular también la bondad del ajuste, es decir la probabilidad de que el chi2 sea mayor que el valor medido si el modelo es correcto, si los datos vienen realmente del modelo teórico estudiado. Queremos de esta manera saber si ajustar el modelo por una recta sin pendiente nos vale o no, de manera precisa.

```r
dof1 <- length(fractionLostByYear)-1
#Se usa lower.tail = FALSE porque estamos buscando valores de chi2 mayores que...
model1Goodness <- pchisq(chi2Model1,dof1,lower.tail = FALSE)
model1Goodness
```

```
## [1] 0.03081716
```

Vemos que en este caso obtenemos un valor de probabilid pequeño, pero no cerca de 0, lo que significa que nuestros datos se parecen bastante al modelo teórico usado pero no podemos concluir que los datos que tenemos provienen (o no) del modelo constante que acabamos de calcular.

En resumen, en esta parte hemos intentado ajustar nuestros datos a una constante (y hemos visto que la constante que mejor aproxima los datos vale 0.2115, de dos maneras diferentes). Hemos calculado también el valor del chi cuadrado en este caso, que vale 34.652 ± 0.017.

Ajuste a una recta

Ahora repetimos el mismo proceso, pero ajustando nuestros datos a una recta que tenga una pendiente en principio no nula.

```r
secondModelCoef <- lm(fractionLostByYear ~ years, weights = 1/estimatedError^2)
intercept <- secondModelCoef$coefficients[1]
intercept
```

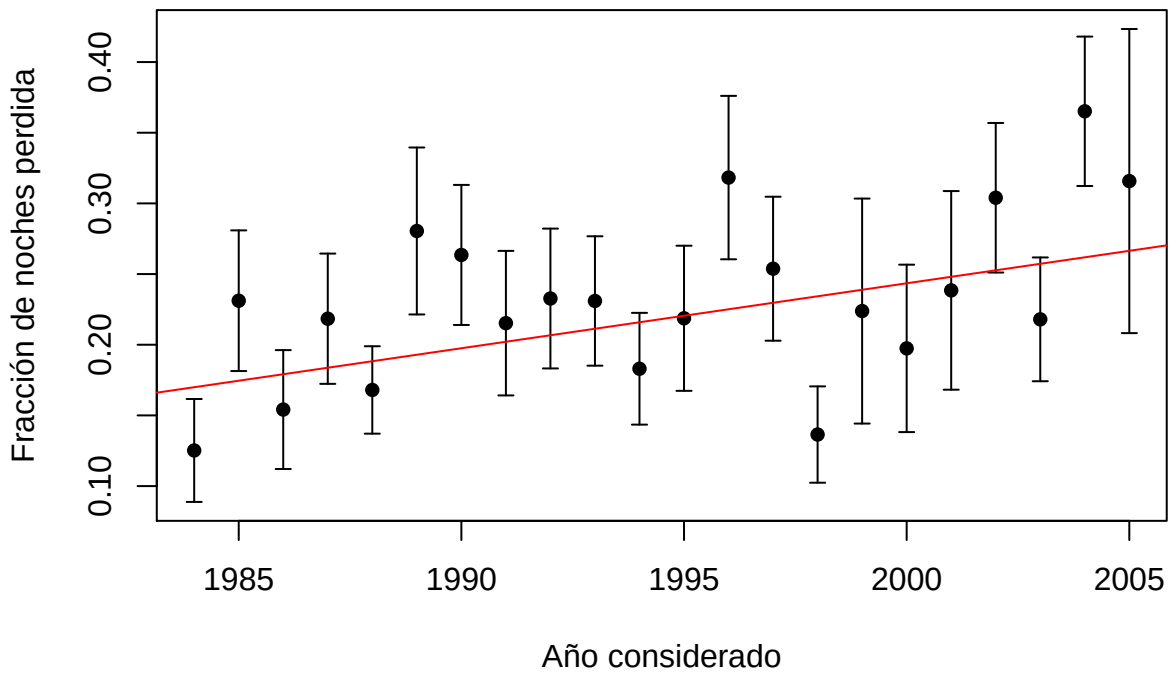
```
## (Intercept)
##   -8.946365
```

```r
slope <- secondModelCoef$coefficients[2]
slope
```

```
##       years
## 0.004594915
```

En este caso vemos que la mejor recta de este tipo que ajusta nuestros datos tiene por ecuación 0.004594915x - 8.946365. Volvemos ahora a representar nuestros datos y la recta que acabamos de calcular en el mismo plot.

```r
errbar(years, fractionLostByYear,
       yplus = fractionLostByYear + estimatedError, ymin = fractionLostByYear - estimatedError,
       xlab = "Año considerado", ylab = "Fracción de noches perdida")
title(main = "Proporción de noches perdidas por año")
abline(a=intercept, b=slope, col="red")
```



Podemos también volver a calcular el chi cuadrado, que depiende ahora de dos parámetros. Lo que hacemos entonces, como tenemos unos conjuntos de puntos (a, b) en el espacio plano es calcular chi2 con la función *optim*, porque esta función nos devuelve los parámetros a (pendiente) y b (termino indepediente) que minimizan el valor de chi2. Por supuesto, el método tiene que devolvernos como valor de a el valor que encotramos antes de 0.004594915, y el valor de b tiene que ser igual a -8.946365.

```
#Función que tendremos que minimizar
functionToMinimize <- function(datay, datax, parameters, error) {
  return(getChi2(datay, parameters[1]+(parameters[2]*datax), estimatedError))
}
#Evaluación de los nuevos parámetros y cálculo del valor de chi2
solution <- optim(par = c(intercept,slope),
                  fn = functionToMinimize, datay=fractionLostByYear, datax=years,
                  error=estimatedError, hessian=TRUE)
newIntercept <- solution$par[1]
newIntercept
```

```
## (Intercept)
##   -8.946365
```

```
newSlope <- solution$par[2]
newSlope
```

```
##       years
## 0.004594915
```

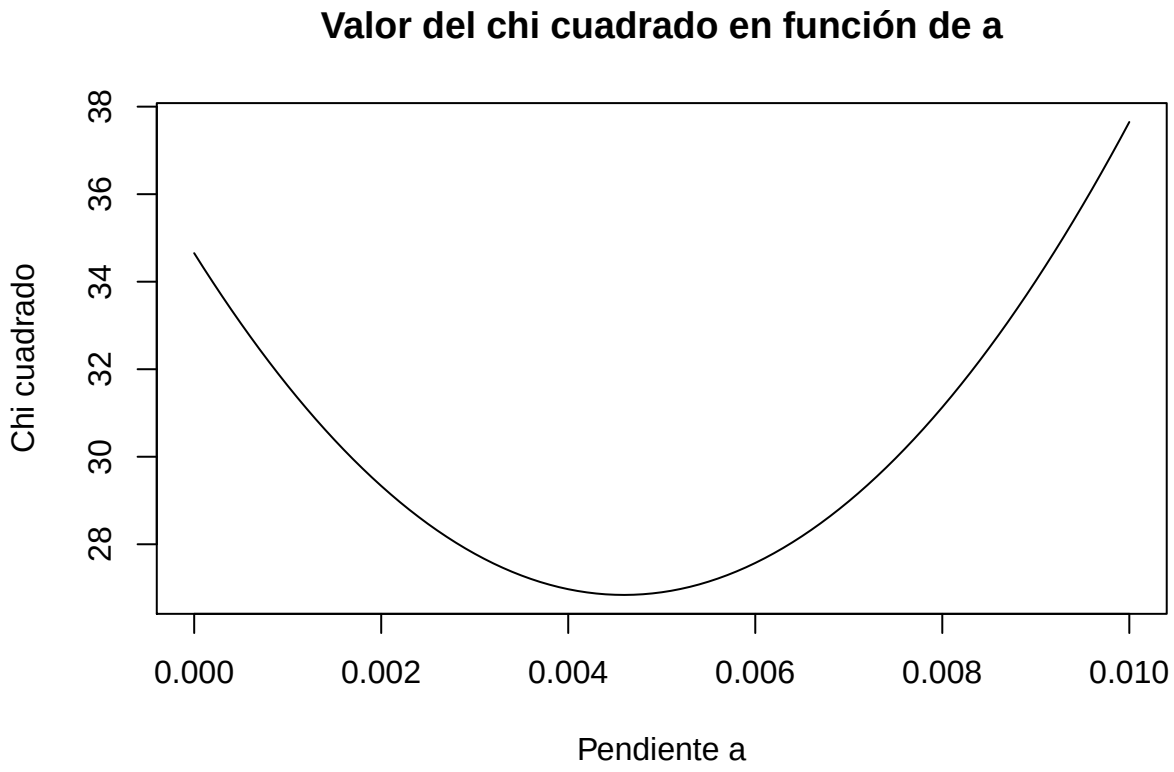
```
chi2Model2 <- solution$value
chi2Model2
```

```
## [1] 26.84252
```

Vemos que con este método, volvemos a obtener exactamente los mismos parámetros (pendiente, termino indepediente) que antes para nuestra recta, lo que nos confirma el método usado. Vemos también que obtenemos un valor de chi2 de 26.92, un poco más pequeño que en el caso precediente (y esto es lo que esperabamos, porque en este caso el ajuste se hace con un parámetro adicional y tiene que ser mejor que el ajuste con una constante).

Ahora podemos volver a calcular el intervalo de confianza del 90% en este caso también. Los intervalos de confianza se calculan primero con las expresiones de los apuntes, y después con la matriz hessiana. El primer proceso usado ya no es tan fácil que en el caso precediente, porque el chi2 depende ahora de dos parámetros. Vamos a ir dando valores a la pendiente a. Para cada valor, tenemos la funcion chi2 que solo depende de b y la minimizamos para encontrar el termino independiente b. Con este último valor y el valor de a, podemos volver a calcular el chi2 que entra en la gráfica. Para calcular el valor del intervalo de confianza del 90%, tendremos que encontrar los valores de parámetros que corresponden al valor minimo de chi2 que aparece en la gráfica, más 2,71.

```
functionToMinimize2 <- function(datay, datax, intercept, slope, error) {
  return(getChi2(datay, intercept+(slope*datax), estimatedError))
}

a <- seq(0, 0.01, 0.0001)
b <- c()
chi2Model2 <- c()

for(i in 1:length(a)) {
  result <- optimize(functionToMinimize2,
                     datay = fractionLostByYear, datax = years, error=estimatedError,
                     slope=a[i], interval=c(-30, 10), tol = 1e-6)
  b[i] <- result$minimum
  chi2Model2[i] <- getChi2(fractionLostByYear, result$minimum+(a[i]*years), estimatedError)
}

plot(a,chi2Model2,type="l", main="Valor del chi cuadrado en función de a",
     xlab="Pendiente a", ylab="Chi cuadrado")
```



Ahora podemos calcular los parámetros a y b que corresponden al minimo de la curva de chi2.

```
indexOptimal <- which.min(chi2Model2)
aOptimal <- a[indexOptimal]
aOptimal
```

```
## [1] 0.0046
```

```
bOptimal <- b[indexOptimal]
bOptimal
```

```
## [1] -8.956501
```

```
chi2Minimum <- chi2Model2[indexOptimal]
chi2Minimum
```

```
## [1] 26.84253
```

Vemos que volvemos a obtener casi exactamente los mismos parámetros que los que salen de la función lm, y el mismo valor del minimo de chi2, usando un método diferente. Podemos ahora calcular el intervalo de confianza del 90%.

```
chi2Model2Lower <- chi2Model2[1:(indexOptimal-1)]
chi2Model2Upper <- chi2Model2[(indexOptimal+1):length(chi2Model2)]
aLower <- a[1:(indexOptimal-1)]
aUpper <- a[(indexOptimal+1):length(a)]
bLower <- b[1:(indexOptimal-1)]
```

```r
bUpper <- b[(indexOptimal+1):length(b)]

aLower[which.min(abs(chi2Model2Lower - (chi2Minimum + 2.71)))]
```

```
## [1] 0.0019
```

```r
aUpper[which.min(abs(chi2Model2Upper - (chi2Minimum + 2.71)))]
```

```
## [1] 0.0073
```

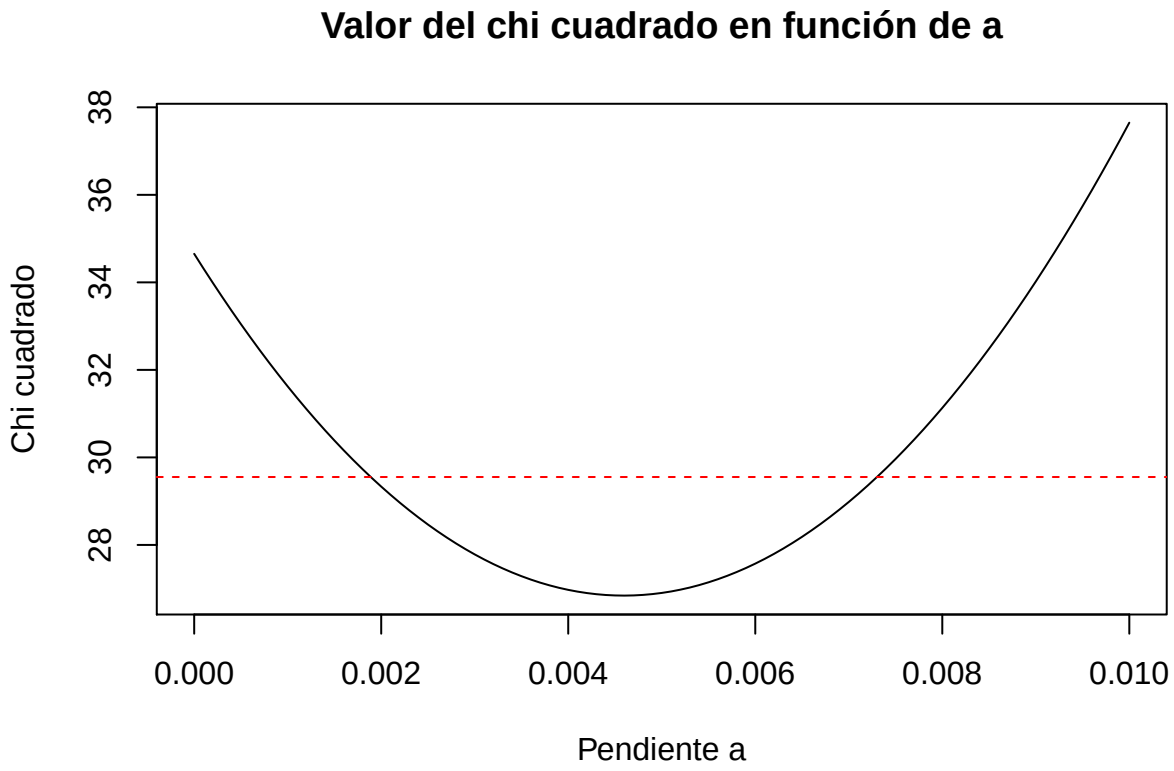
```r
bLower[which.min(abs(chi2Model2Lower - (chi2Minimum + 2.71)))]
```

```
## [1] -3.575252
```

```r
bUpper[which.min(abs(chi2Model2Upper - (chi2Minimum + 2.71)))]
```

```
## [1] -14.33775
```

```r
plot(a,chi2Model2,type="l", main="Valor del chi cuadrado en función de a",
     xlab="Pendiente a", ylab="Chi cuadrado")
abline(h=chi2Minimum+2.71, col="red", lty=2)
```



Podemos entonces afirmar con una certitud del 90% que la pendiente tiene que estar en el intervalo [0.0019, 0.0073] y que el termino independiente tiene que estar en el intervalo [-3.5752, -14.3377].

Comparación de los ajustes

Ahora que hemos calculado muchos parámetros con dos ajustes diferentes, podemos compararlos con un simple test F. Nos esperamos a ver que el segundo ajuste sea mejor (valor de probabilidad nula menor), porque tiene un parámetro de ajuste aditional.

```
#Test F sobre los dos modelos según la formula vista en clase
dof1 <- length(fractionLostByYear)-1
dof2 <- length(fractionLostByYear)-2
statisticsF <- ((min(chi2Model1)^2 - min(chi2Model2)^2)/(dof1 - dof2))/(min(chi2Model2)/dof2)
as.numeric(pf(statisticsF, dof1-dof2, dof2))
```

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

Vemos que el número que sale de la función pf vale exactamente 1. Eso nos indica que la probablidad nula es pequeña (1-el valor obtenido por 100% nos devuelve 0), y que por lo tanto nuestro segundo modelo es mejor que el primero, como lo esperebamos.

Segunda parte

La segunda parte del ejercicio consiste primero a repetir lo mismo (ajuste a una constante), pero con nuevos datos. Los datos que usamos en esta parte son la fracción de noches perdidas por mes, y nor por año como en la primera parte del ejercicio. Primero, creamos la nueva tabla que usaremos.

```
fractionLostByMonth <- c()
estimatedError <- c()
months <- c("January", "February", "March", "April", "May", "June",
            "July", "August", "September", "October", "November", "December")
fractionLostByMonth <- colMeans(fractionLost, na.rm = T)
for(i in 1:12) {
  estimatedError[i] <- sd(fractionLost[,i], na.rm = T)/sqrt(length(which(!is.na(data[,i]))))
}
newData <- data.frame(months, fractionLostByMonth, estimatedError)
newData
```

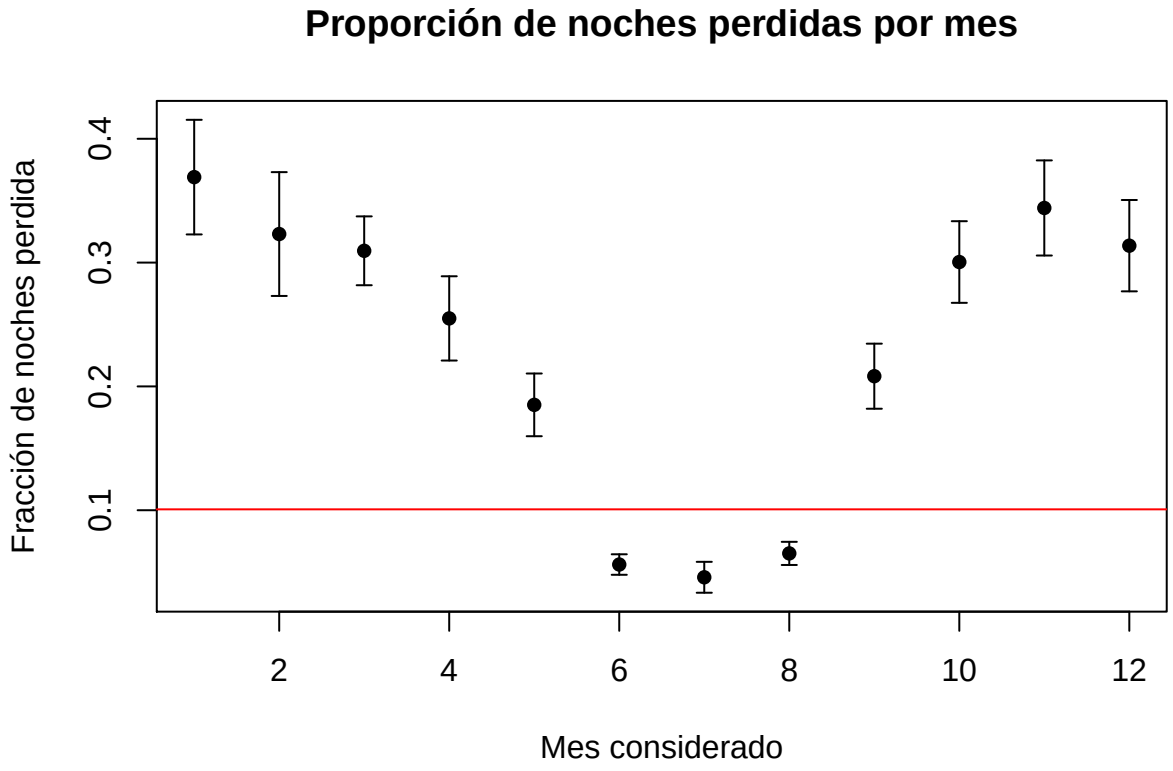
```
##        months fractionLostByMonth estimatedError
## Ene   January          0.36903226    0.046285072
## Feb  February          0.32307061    0.050001081
## Mar     March          0.30952381    0.027806944
## Abr     April          0.25492063    0.034031280
## May       May          0.18509985    0.025319402
## Jun      June          0.05614035    0.008280586
## Jul      July          0.04584041    0.012472969
## Ago    August          0.06513057    0.009350916
## Sep September          0.20825397    0.026277537
## Oct   October          0.30046083    0.032964076
## Nov  November          0.34412698    0.038412206
## Dic  December          0.31367127    0.036888754
```

Ajustamos ahora una constante a nuestros datos como lo hicimos antes, y pintamos todo.

```
secondModelCoef <- lm(fractionLostByMonth ~ 1, weights = 1/estimatedError^2)
secondModelCoef$coefficients
```

```
## (Intercept)
##   0.1007219
```

```
errbar(x = 1:12, y = fractionLostByMonth, yplus = fractionLostByMonth + estimatedError, ymin = fractionL
title(main = "Proporción de noches perdidas por mes")
abline(h=secondModelCoef$coefficients, col="red")
```



Vemos que en este caso, claramente ajustar nuestros datos por un modelo constante no tiene sentido. Algunos puntos están muy lejos (hasta incluso más que 4 o 5 sigmas) de la distribución teórica constante.

```
chi2Part2 <- getChi2(fractionLostByMonth, secondModelCoef$coefficients, estimatedError)
chi2Part2
```

```
## [1] 331.1724
```

```
error90Chi2Part2 <- sqrt(2.71/sum(1/estimatedError^2))
error90Chi2Part2
```

```
## [1] 0.008148668
```

```
dof1Part2 <- 11
#Se usa lower.tail = FALSE porque estamos buscando valores de chi2 mayores que...
part2Goodness <- pchisq(chi2Part2, dof1Part2, lower.tail = FALSE)
part2Goodness
```

```
## [1] 2.319959e-64
```

Obtenemos un valor de chi2 mucho más alto que en la primera parte, y un valor de bondad muy pequeño y igual a 0 lo que nos significa que nuestro modelo no es bueno.

Ahora podemos intentar modelizar nuestros datos con otro modelo de tipo périodico, que tiene 3 parámetros libres. En este caso, volvemos a tener una situación parecida a lo que hicimos en el caso de la recta de pendiente no nula del caso precediente. El modelo que consideramos en este caso consiste en sumar una constante llamada fc (que depiende por ejemplo de la localisación del observador) por una variación periodica (de periodo 12 meses) que depende de dos parámetros : una fase t0 y fe, un parámetro que multiplica el seno. La función que vamos a mirar tiene entonces la forma siguiente.

```
periodicModel <- function (data, t0, fc, fe) {
  output <- fc + fe*sin(2*3.1415*(data-t0)/12)
  return(output)
}
```

Primero, calculamos y escribimos la función que queremos minimizar.

```
# Función que queremos minimizar
periodicFunctionToMinimize <- function(parameter, x, y, error) {
  output <- getChi2(y, periodicModel(x, parameter[1], parameter[2], parameter[3]), error)
  return(output)
}
```

Ahora, calculamos el valor de los nuevos parámetros para que minimizen el valor de chi2, por el uso de la función optim.

```
#Valores iniciales de los parámetros
t0 <- 1
fc <- 1
fe <- 1

newParameters <- optim(c(t0,fc,fe), periodicFunctionToMinimize,
                       x=c(1:12), y=fractionLostByMonth,
                       error=estimatedError, hessian=TRUE)
newT0 <- newParameters$par[1]
newT0
```

```
## [1] 3.871763
```

```
newFc <- newParameters$par[2]
newFc
```

```
## [1] 0.2262365
```

```
newFe <- newParameters$par[3]
newFe
```

```
## [1] -0.182183
```

```r
periodicChi2 <- newParameters$value
periodicChi2
```

```
## [1] 24.37156
```

```r
#Volvemos a repetir dos veces el proceso con otros parámetros iniciales para evitar problema de minimo
t0 <- -5
fc <- 10
fe <- 0.1

newParameters <- optim(c(t0,fc,fe), periodicFunctionToMinimize,
                       x=c(1:12), y=fractionLostByMonth,
                       error=estimatedError, hessian=TRUE)
newT0 <- newParameters$par[1]
newT0
```

```
## [1] 3.862827
```

```r
newFc <- newParameters$par[2]
newFc
```

```
## [1] 0.2259495
```

```r
newFe <- newParameters$par[3]
newFe
```

```
## [1] -0.1822584
```

```r
periodicChi2 <- newParameters$value
periodicChi2
```

```
## [1] 24.38419
```

```r
t0 <- 5
fc <- -2
fe <- -3

newParameters <- optim(c(t0,fc,fe), periodicFunctionToMinimize,
                       x=c(1:12), y=fractionLostByMonth,
                       error=estimatedError, hessian=TRUE)
newT0 <- newParameters$par[1]
newT0
```

```
## [1] 3.864816
```

```r
newFc <- newParameters$par[2]
newFc
```

```
## [1] 0.2253479
```

```r
newFe <- newParameters$par[3]
newFe
```

```
## [1] -0.1814698
```

```r
periodicChi2 <- newParameters$value
periodicChi2
```

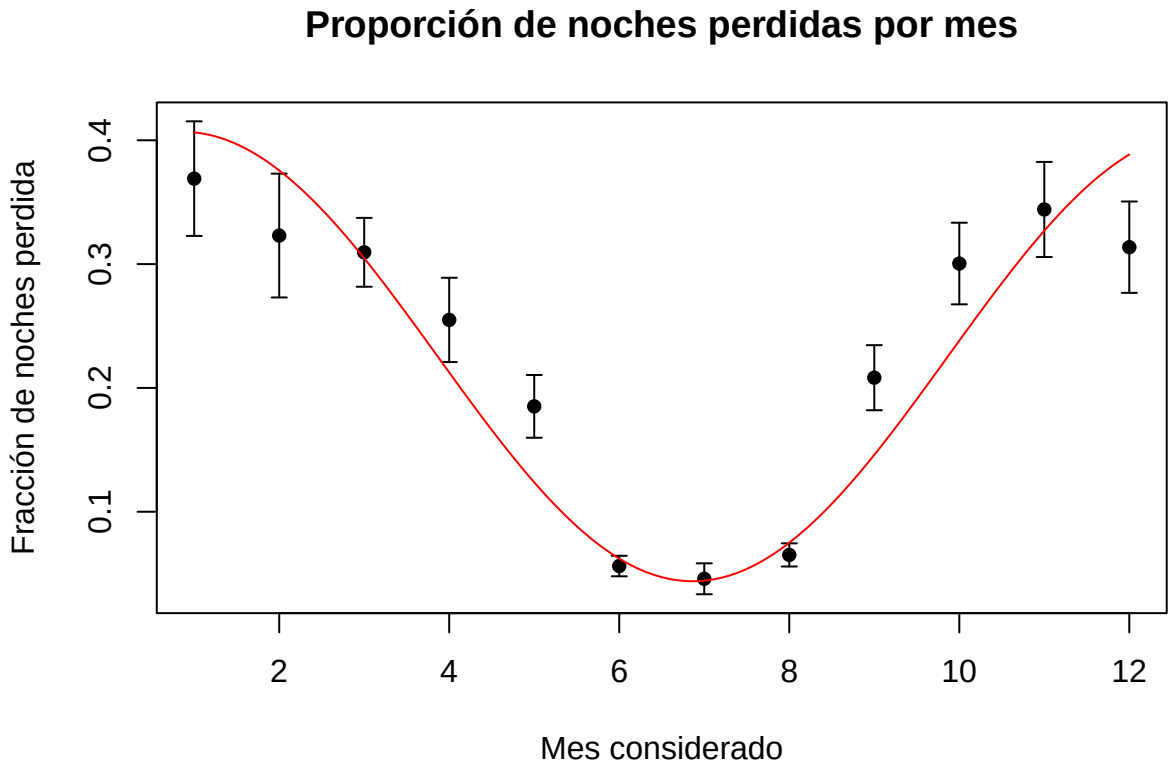
```
## [1] 24.38447
```

```r
#Cálculo de la bondad
part2Goodness2<- pchisq(periodicChi2, dof1Part2, lower.tail = FALSE)
part2Goodness2
```

```
## [1] 0.01120616
```

Con el valor de bondad que obtenemos, no se pueda afirmar entonces que el modelo no es consistente con los datos. Ahora que tenemos todos los parámetros de nuestro modelo (y que hemos verificado un poco que no tenemos un minimo local, aunque sería mejor verificarlo con muchos parámetros iniciales y no solamentes tres combinaciones), solo hace falta pintar todo para ver si este nuevo modelo queda bien, o no.

```r
errbar(x = 1:12, y = fractionLostByMonth, yplus = fractionLostByMonth + estimatedError, ymin = fractionL
title(main = "Proporción de noches perdidas por mes")

sinusoidalFit <- periodicModel(seq(1,12,0.01), newT0, newFc, newFe)
lines(seq(1,12,0.01), sinusoidalFit, col="red")
```

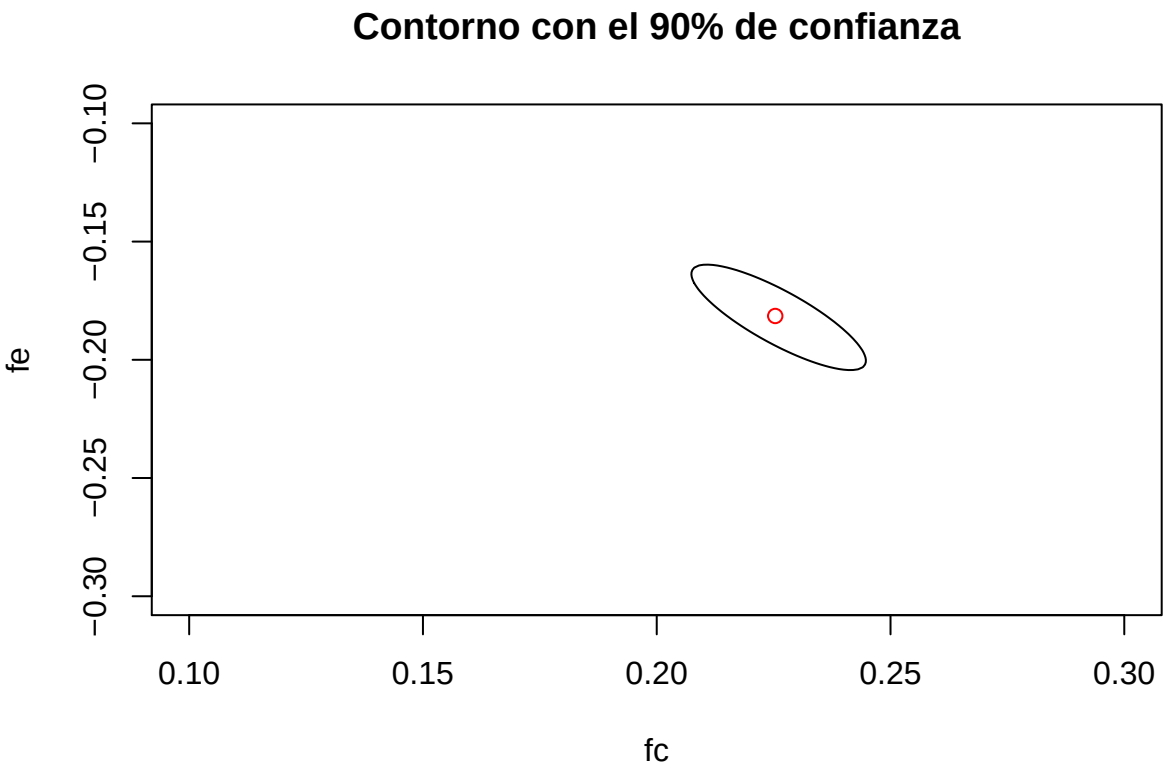


En este caso, se ve que el modelo periodico introducido queda bien en comparación con los datos experimentales (no hay puntos que están más lejos de 2 sigma de la distribución teórica). Solo hace falta ahora calcular el error sobre los parámetros que acabamos de calcular (en este caso, el intervalo de confianza del 90% es un contorno en dos dimensiones, porque tenemos tres parámetros a nuestro modelo). Lo que hacemos entonces es fijar diferentes valores de los parámetros fc y fe, para ver qué valor de t0 nos permite minimizar el valor del chi cuadrado, un poco como lo hicimos en la primera parte del ejercicio.

```
variationFc <- seq(0.1,0.3,0.001)
variationFe <- seq(-0.3,-0.1,0.001)
chi2Matrix <- matrix(rep(NA,10050),
                     nrow=length(variationFc), ncol=length(variationFe), byrow=TRUE)

functionToMinimize3 <- function(parameter, x, y, error, fc, fe) {
  t0 <- parameter[1]
  output <- getChi2(y, periodicModel(x,t0,fc,fe), error)
  return(output)
}

# Recorremos todos los posibles valores de fc y fe
for (i in 1:length(variationFc)) {
  for (j in 1:length(variationFe)) {
    optimization <- optimize(functionToMinimize3, lower=0, upper=13, x=c(1:12),
                             y=fractionLostByMonth, error = estimatedError,
                             fc=variationFc[i], fe=variationFe[j])
    chi2Matrix[i,j] <- optimization$objective
  }
}

#Calculamos el minimo de la matriz
chi2MatrixMin <- min(chi2Matrix)
chi2MatrixMin
```

```
## [1] 24.37123
```

```
#Pintamos el contorno
contour(variationFc, variationFe, chi2Matrix, levels=c(chi2MatrixMin + 4.61),
        xlab="fc", ylab="fe", main="Contorno con el 90% de confianza", drawlabels = F)
#Dibujamos el valor mínimo de chi cuadrado como un punto en el contorno
points(newFc,newFe, col="red")
```



Conclusión

En la primera parte del ejercicio, intentamos ajutar nuestros datos (número de noches de observación astronómica perdidas por culpa del tiempo, por mes y por año) a dos modelos diferentes. Primero, a un modelo constante (calculamos de dos maneras diferentes que la constante que modeliza mejor la proporción de noches perdida por año vale 0.2116 y vimos que este modelo nos devuelve un valor de chi cuadrado de 34.652 ± 0.017). Después, intentamos ajustar estos datos a un modelo que tenga un segundo parámetro libre (une recta, que tiene su pendiente y su termino independiente). En este caso, vimos que la recta que aproxima mejor nuestros datos tiene por ecuación 0.004594915x - 8.946365 y un valor de chi cuadrado un poco menor que en el caso anterior. Hicimos también un test F para poder comparar estos dos modelos y concluimos con esta prueba que el segundo modelo está mejor que el primero (y es lo que esperebamos, porque tiene un parámetro libre más).

En la segunda parte del ejercicio, estudiamos la fracción de noches perdidas por mes, y no por año. En este caso, intentamos ajustar también un modelo constante (que vale 0.1007) y vimos gráficamente que este modelo no es bueno para explicar nuestros datos (valor de chi cuadrado muy alto, mayor que 300). Intentamos entonces ajustar un modelo de tipo periodico a los datos, con el uso de tres parámetros (la fase t0, una constante fc y un parámetro fe que multiplica un seno). Vimos en este caso que los datos se approximan mucho más al modelo teórico. El intervalo de confianza del 90% en los nuevos parámetros también ha sido calculado, en la forma de un contorno bidimensional.

Bibliografia

R Development Core Team (2008). *R: A language and environment for statistical computing. R Foundation for Statistical Computing*, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

R Markdown, *Markdown basics*, http://rmarkdown.rstudio.com/authoring_basics.html. Consultado por última vez el 14 de noviembre 2016.

Find out the number of days of a month in R, Stackoverflow, http://stackoverflow.com/questions/6243088/find-out-the-number-of-days-of-a-month-in-r. Consultado por última vez el 28 de octubre 2016.

Get Number of Days in Year or Month, Ecole Polytechnique Federale de Lausanne, http://svitsrv25.epfl.ch/R-doc/library/Hmisc/html/yearDays.html. Consultado por última vez el 28 de octubre 2016.

Data Frame, R tutorial, http://www.r-tutor.com/r-introduction/data-frame. Consultado por última vez el 10 de noviembre 2016.

How to use optim in R, Magesblog, http://www.magesblog.com/2013/03/how-to-use-optim-in-r.html. Consultado por última vez el 16 de noviembre 2016.

Find the closest value in a list or matrix, https://stat.ethz.ch/pipermail/r-help/2008-July/167216.html. Consultado por última vez el 16 de noviembre 2016.