UNIVERSITAT DE BARCELONA

Universitat Oberta de Catalunya

Regresión segmentada

Francesc Carmona

13 de diciembre de 2023

```
if(!(require(strucchange))) install.packages("strucchange")
if(!(require(minpack.lm))) install.packages("minpack.lm")
if(!(require(rcompanion))) install.packages("rcompanion")
if(!(require(nlstools))) install.packages("nlstools")
if(!(require(segmented))) install.packages("segmented")
```

Introducción

La regresión segmentada o regresión a trozos es un método de regresión en que la variable predictora se divide en intervalos de forma que se ajusta una línea o curva a los datos en cada intervalo. La regresión segmentada es útil cuando la variable respuesta muestra una reacción abruptamente diferente a la variable predictora en los diferentes segmentos. En este caso el límite entre los segmentos se llama punto de quiebra. La regresión segmentada se puede aplicar también a la regresión con múltiples variables predictoras. En este documento trataremos únicamente el caso de la regresión lineal simple (una única variable explicativa) segmentada en dos o más trozos.

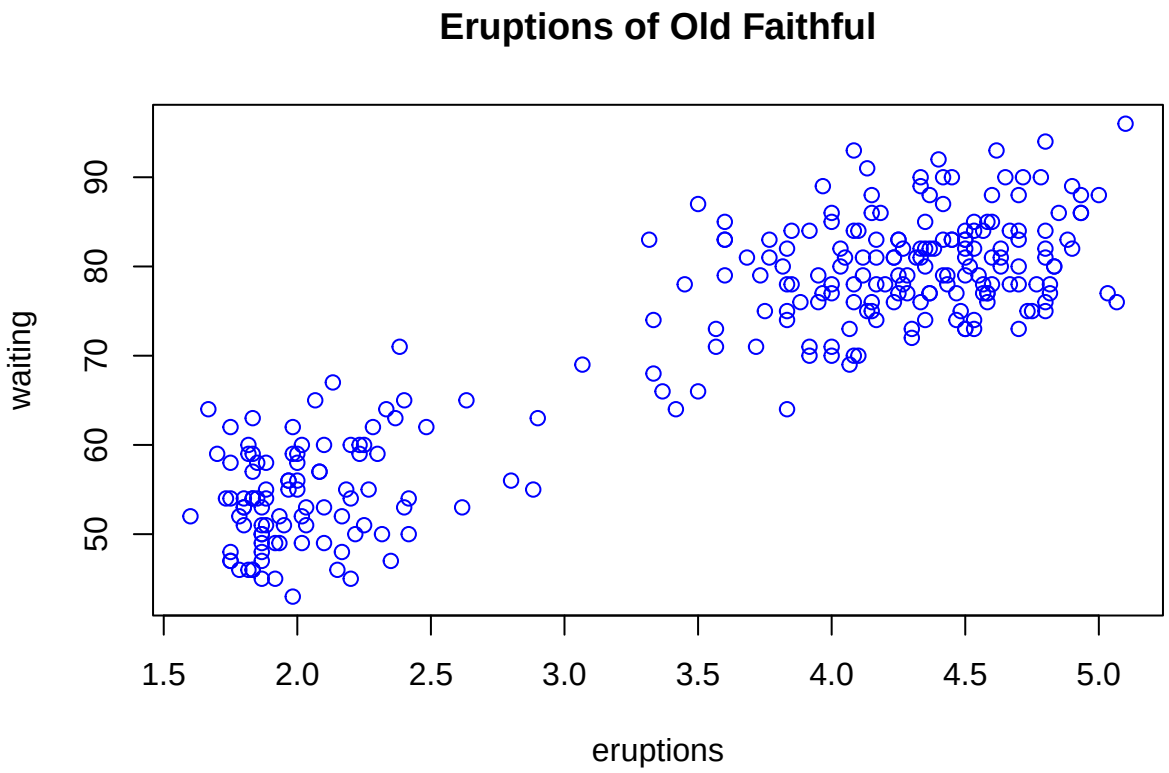
Para ello vamos a utilizar los famosos datos del geyser Old Faithful del parque de Yellowstone.

```
of <- faithful
str(of)
```

```
## 'data.frame':    272 obs. of  2 variables:
##  $ eruptions: num  3.6 1.8 3.33 2.28 4.53 ...
##  $ waiting  : num  79 54 74 62 85 55 88 85 51 85 ...
```

Se trata de algunos datos recogidos en los años 80 sobre la duración de la erupción (en minutos) y el tiempo de espera (en minutos) hasta la siguiente.

```
plot(of, col="blue",main="Eruptions of Old Faithful")
```



El gráfico muestra dos grupos de datos según la duración de forma que nos podemos plantear si es necesaria una regresión segmentada.

Test de Chow

El test de Chow se utiliza para decidir si tiene sentido hacer dos regresiones separadas para dos intervalos de los datos divididos en un punto de quiebra c (*breakpoint*).

Se trata de un test de coincidencia entre las rectas de regresión de los dos intervalos por separado

$$y_i = \alpha_1 + \beta_1 x_i + \epsilon_i, \quad \text{para } x_i < c$$
$$y_i = \alpha_2 + \beta_2 x_i + \epsilon_i, \quad \text{para } x_i \geq c$$

con la recta de regresión conjunta, es decir,

$$H_0 : \alpha_1 = \alpha_2,\ \beta_1 = \beta_2$$

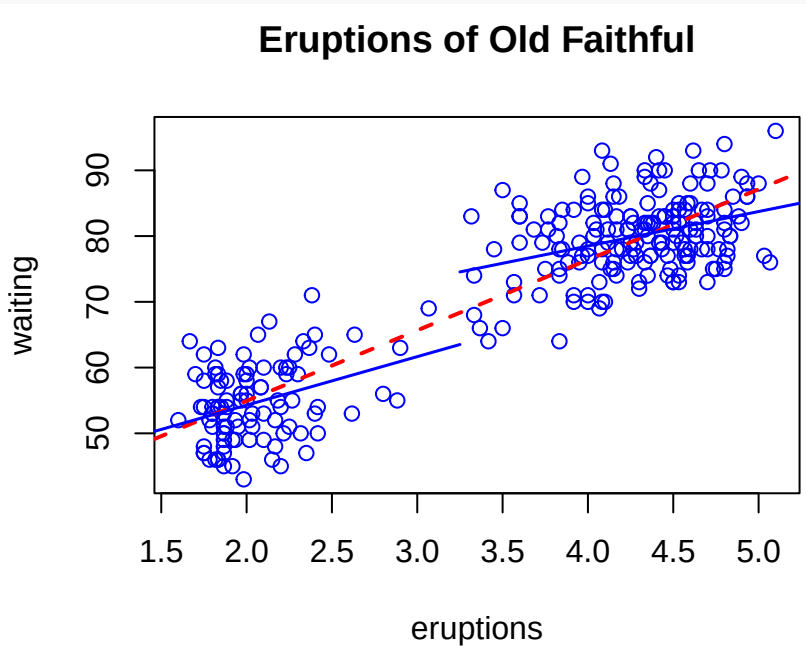
Es un test F que se puede calcular con las sumas de cuadrados residuales de las rectas de regresión: dos para los dos intervalos por separado y la conjunta (ver referencia). Sin embargo, en R podemos plantear un contraste de modelos con la creación de una variable auxiliar dicotómica. A la vista del gráfico de dispersión parece que el punto de corte entre los dos grupos ocurre en el valor de $c = 3.25$ minutos. Tomaremos este valor como punto de quiebra.

```
dummy <- as.numeric(of$eruptions >= 3.25)
lmod <- lm(waiting ~ eruptions * dummy, data = of)
lmod0 <- lm(waiting ~ eruptions, data = of)
anova(lmod0, lmod)
```

```
## Analysis of Variance Table
##
## Model 1: waiting ~ eruptions
## Model 2: waiting ~ eruptions * dummy
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1    270 9443.4
## 2    268 8416.3  2      1027 16.352 1.993e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

El resultado es que rechazamos el modelo más simple con la recta conjunta. Podemos hacer un gráfico con los resultados.

```
plot(of, col="blue", main="Eruptions of Old Faithful")
abline(lmod0, col = "red", lwd = 2, lty = "dashed")
segments(0.00, lmod$coef[1],
         3.25, lmod$coef[1]+3.25*lmod$coef[2], col= 'blue', lwd=1.5)
# unrestricted model 2
segments(3.25, (lmod$coef[1]+lmod$coef[3])+3.25*(lmod$coef[2]+lmod$coef[4]),
         5.50, (lmod$coef[1]+lmod$coef[3])+5.50*(lmod$coef[2]+lmod$coef[4]),
         col= 'blue', lwd=1.5)
```



El mismo test se puede obtener con la función `sctest()` del paquete `strucchange`. Sin embargo, esta solución requiere que identifiquemos la fila que corresponde al punto de quiebra, lo que resulta un poco más complicado.

```
library(strucchange)
# Sort the data
sort.of <- of[order(of$eruptions), ]
sort.of <- cbind(index(sort.of),sort.of)
# Identify the row number of our breakpoint
```

```r
brk <- which(sort.of$eruptions == max(sort.of$eruptions[sort.of$eruptions < 3.25]))
# Using the 'sctest()' function
sctest(waiting ~ eruptions, type = "Chow", point = brk, data = sort.of)
```

```
##
##  Chow test
##
## data:  waiting ~ eruptions
## F = 16.352, p-value = 1.993e-07
```

Como se aprecia en el gráfico anterior, entre los segmentos de los dos intervalos no hay continuidad. En el siguiente apartado vamos a demandar esa continuidad.

Broken Stick Regression

En el apartado 9.3 del libro de Faraway (pág. 137) se explica como crear dos funciones llamadas bases para formar una regresión a trozos con continuidad.

$$B_l(x) = \begin{cases} c - x & \text{si } x < c \\ 0 & \text{en otro caso} \end{cases}$$

y

$$B_r(x) = \begin{cases} x - c & \text{si } x > c \\ 0 & \text{en otro caso} \end{cases}$$

donde c es el punto de quiebra.

Con estas funciones ajustamos un modelo en la forma

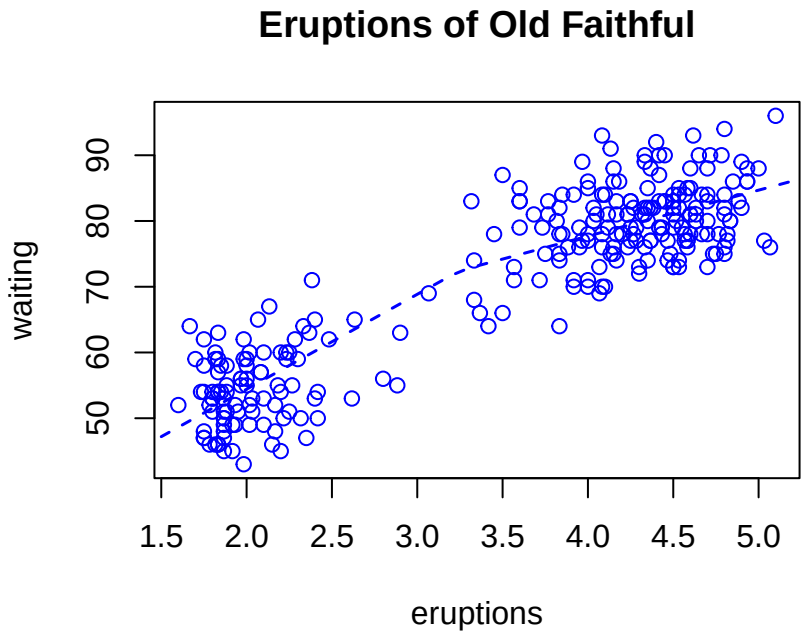
$$y = \beta_0 + \beta_1 B_l(x) + \beta_2 B_r(x) + \epsilon$$

Así se garantiza que los dos segmentos se unen en el punto c.

Debemos notar que este modelo utiliza únicamente tres parámetros en contraste con el del apartado anterior. Uno de los parámetros se ha eliminado para obtener la continuidad en c.

Con R los cálculos son:

```r
lhs <- function(x) ifelse(x < 3.25,3.25-x,0)
rhs <- function(x) ifelse(x < 3.25,0,x-3.25)
lmsr <- lm(waiting ~ lhs(eruptions) + rhs(eruptions), of)
x <- seq(1.5,5.5,by=0.1)
py <- lmsr$coef[1] + lmsr$coef[2]*lhs(x) + lmsr$coef[3]*rhs(x)
plot(of, col="blue", main="Eruptions of Old Faithful")
lines(x, py, lty=2, lwd=1.5, col="blue")
```



También hay otra forma de conseguir la misma regresión a trozos utilizando la variable auxiliar dicotómica (ver Piecewise Linear Regression Models del curso STAT 501 de PennState).

```
y <- of$waiting
x1 <- of$eruptions
# dummy
x2 <- dummy # as.numeric(x1 >= 3.25)
# interaction
x2star <- (x1 - 3.25) * x2
lmod2 <- lm(y ~ x1 + x2star)
summary(lmod2)
```

```
##
## Call:
## lm(formula = y ~ x1 + x2star)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -12.5330  -4.0747   0.3533   3.5523  14.7196
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  25.5418     2.3054  11.079  < 2e-16 ***
## x1           14.4357     0.9889  14.598  < 2e-16 ***
## x2star       -7.4460     1.8889  -3.942 0.000103 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.761 on 269 degrees of freedom
## Multiple R-squared:  0.8218, Adjusted R-squared:  0.8204
```

```
## F-statistic: 620.1 on 2 and 269 DF,  p-value: < 2.2e-16
```

Los coeficientes coinciden con los del modelo de regresión anterior:

```
c(lmod2$coef[2], -lmsr$coef[2])
```

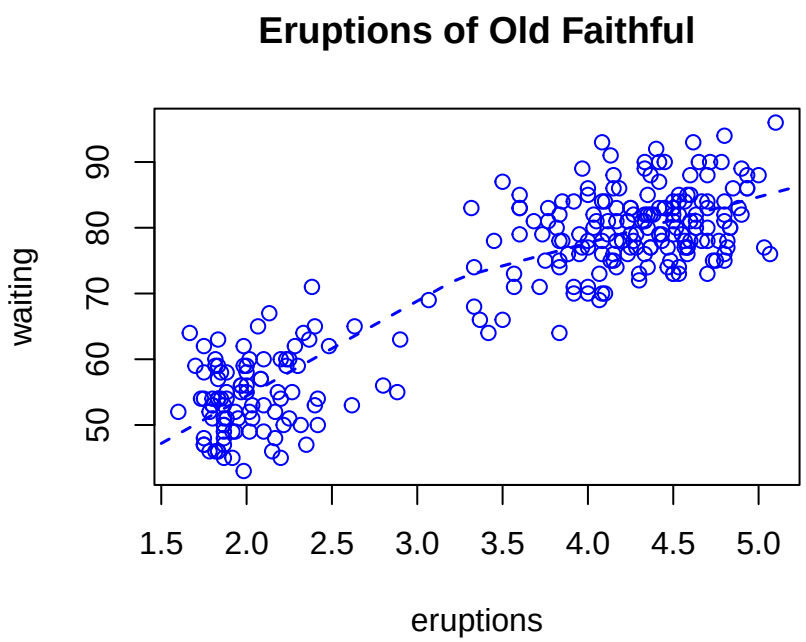
```
##             x1 lhs(eruptions)
##       14.43575       14.43575
```

```
c(lmod2$coef[2] + lmod2$coef[3], lmsr$coef[3])
```

```
##             x1 rhs(eruptions)
##       6.989713       6.989713
```

Finalmente podemos repetir el gráfico una vez más.

```
plot(of, col="blue", main="Eruptions of Old Faithful")
myplotgrid <- seq(1.5, 5.5, by=0.1)
my.predicted.values <-
  predict(lmod2,
  newdata=data.frame(x1 = myplotgrid,
                     x2star = (myplotgrid - 3.25)*as.numeric(myplotgrid > 3.25)))
lines(myplotgrid, my.predicted.values, col="blue", lty=2, lwd=1.5)
```



Punto de corte óptimo

Hay varias formas de hallar el punto de corte óptimo cuando el punto de quiebra es desconocido. En este caso vamos a utilizar una función que permite hallar ese punto optimizando la suma de cuadrados residual.

```
pick.changepoint <- function(response, predictor,
                             min.changept, max.changept, gridlength){
  my.changept.choices <- pretty(c(min.changept, max.changept), n=gridlength)
  results <- matrix(0, nrow=length(my.changept.choices), ncol=2)
```

```
  for (i in 1:length(my.changept.choices))
  {
    x2star <-
     (predictor - my.changept.choices[i])*as.numeric(predictor > my.changept.choices[i])
    fit <- lm(response ~ predictor + x2star)
    results[i,1] <- my.changept.choices[i]
    results[i,2] <- anova(fit)["Residuals", "Sum Sq"]
  }
  print(paste("Optimal changepoint:", results[results[,2]==min(results[,2]),1] ))
  return(results)
}
```

Con esta función y los datos del geyser tenemos:

```
pick.changepoint(response=y, predictor=x1, min.changept=3.0, max.changept=4.0,
                 gridlength=50)
```

```
## [1] "Optimal changepoint: 3.76"
```

```
##       [,1]     [,2]
##  [1,] 3.00 9115.996
##  [2,] 3.02 9100.224
##  [3,] 3.04 9084.562
##  [4,] 3.06 9069.034
...
```

Según este criterio, utilizar un punto de corte $c = 3.76$ produce el menor RSS.

Ahora vamos a probar con la función `nls()` que determina tanto los estimadores mínimo cuadráticos no lineales de los coeficientes como del punto de quiebra. Esta función es mucho más flexible que la anterior y, como veremos, se puede aplicar con otros modelos. Únicamente necesita una función que defina el modelo no lineal y unos valores iniciales de los parámetros a estimar.

La función que nos permitirá definir el modelo es la siguiente:

```
linsegreg <- function(x, beta0, beta1, beta2, cx){
  x1 <- x
  x2 <- as.numeric(x1 >= cx)
  x2star <- (x1 - cx) * x2
  ifelse(x < cx, beta0 + beta1 * x1,
                 beta0 + beta1 * x1 + beta2 * x2star)
}
```

además, tomaremos como valores iniciales los del modelo lineal anterior.

```
model <- nls(waiting ~ linsegreg(eruptions, beta0, beta1, beta2, cx),
             data = of,
             start = list(beta0   = lmod2$coef[1],
                          beta1   = lmod2$coef[2],
                          beta2   = lmod2$coef[3],
                          cx      = 3.25),
             trace = FALSE,
             nls.control(maxiter = 1000))
```

Por desgracia, el algoritmo no converge.

Probaremos pues con la función `nlsLM()` del paquete `minpack.lm`.

```
library(minpack.lm)
model <- nlsLM(waiting ~ linsegreg(eruptions, beta0, beta1, beta2, cx),
               data = of,
               start = list(beta0   = lmod2$coef[1],
                            beta1   = lmod2$coef[2],
                            beta2   = lmod2$coef[3],
                            cx      = 3.25))
summary(model)
```

```
##
## Formula: waiting ~ linsegreg(eruptions, beta0, beta1, beta2, cx)
##
## Parameters:
##                   Estimate Std. Error t value Pr(>|t|)
## beta0.(Intercept)  28.2480     2.0009  14.118  < 2e-16 ***
## beta1.x1           13.0437     0.8379  15.567  < 2e-16 ***
## beta2.x2star       -8.0004     1.6900  -4.734 3.57e-06 ***
## cx                  3.7670     0.2105  17.899  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.694 on 268 degrees of freedom
##
## Number of iterations to convergence: 9
## Achieved convergence tolerance: 1.49e-08
```

El punto de corte óptimo es 3.7670.

Plateau regression

Un problema similar de regresión a trozos se presenta cuando los datos tienen una tendencia al alza (o a la baja) hasta un punto óptimo de estabilización y, a partir de ese punto, el valor de la respuesta es una constante. Se trata pues de un modelo donde la primera parte de los datos se debe ajustar a una regresión lineal, mientras que la segunda parte es constante. Además hay que estimar el punto donde se alcanza el valor constante y dicha constante. Este tipo de problema se conoce como *plateau regression* o regresión con meseta.

En el libro de Mangiafico encontramos el siguiente ejemplo que vamos a reproducir. Los datos están en el apéndice.

El objetivo es crear una función `linplat()` que modele la regresión con meseta para las variables $x =$ *Calories* e $y =$ *Sodium* con los parámetros a o *intercept*, b o *slope* y clx. Este último es el punto de quiebra.

Para estimar estos parámetros utilizaremos la función `nls()` con unos valores iniciales.

```
###  Find reasonable initial values for parameters
fit.lm    <- lm(Sodium ~ Calories, data=Data)
a.ini     <- fit.lm$coefficients[1]
b.ini     <- fit.lm$coefficients[2]
```

```r
clx.ini   <- mean(Data$Calories)

###  Define linear plateau function
linplat <- function(x, a, b, clx)
          {ifelse(x < clx, a + b * x,
                           a + b * clx)}

###  Find best fit parameters
model <- nls(Sodium ~ linplat(Calories, a, b, clx),
               data = Data,
               start = list(a   = a.ini,
                            b   = b.ini,
                            clx = clx.ini),
               trace = FALSE,
               nls.control(maxiter = 1000))
summary(model)
```

```
##
## Formula: Sodium ~ linplat(Calories, a, b, clx)
##
## Parameters:
##       Estimate Std. Error t value Pr(>|t|)
## a   -128.85533  116.10818   -1.11    0.273
## b      0.65768    0.05343   12.31  1.6e-15 ***
## clx 2326.51521   16.82086  138.31  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 24.47 on 42 degrees of freedom
##
## Number of iterations to convergence: 4
## Achieved convergence tolerance: 1.36e-08
```

En esta ocasión el algoritmo funciona y nos proporciona las estimaciones.

Además podemos calcular un coeficiente *pseudo* R^2 como medida de ajuste y un p-valor de significación, si definimos un modelo nulo.

```r
###  Define null model
nullfunct <- function(x, m){m}
m.ini     <- mean(Data$Sodium)
null <- nls(Sodium ~ nullfunct(Calories, m),
              data = Data,
              start = list(m   = m.ini),
              trace = FALSE,
              nls.control(maxiter = 1000))

###  Find p-value and pseudo R-squared
library(rcompanion)
nagelkerke(model, null)
```

```
## $Models
##
## Model: "nls, Sodium ~ linplat(Calories, a, b, clx), Data, list(a = a.ini, b = b.ini, clx = clx.ini)
## Null:  "nls, Sodium ~ nullfunct(Calories, m), Data, list(m = m.ini), list(1000, 1e-05, 0.0009765625
##
## $Pseudo.R.squared.for.model.vs.null
##                              Pseudo.R.squared
## McFadden                             0.195417
## Cox and Snell (ML)                   0.892004
## Nagelkerke (Cragg and Uhler)         0.892014
##
## $Likelihood.ratio.test
##  Df.diff LogLik.diff  Chisq   p.value
##       -2     -50.077 100.15 1.785e-22
##
## $Number.of.observations
##
## Model: 45
## Null:  45
##
## $Messages
## [1] "Note: For models fit with REML, these statistics are based on refitting with ML"
##
## $Warnings
## [1] "None"
```

También se pueden calcular los intervalos de confianza de los parámetros.

```
library(nlstools)
confint2(model, level = 0.95)
```

```
##            2.5 %       97.5 %
## a   -363.1711326  105.4604719
## b      0.5498523    0.7654999
## clx 2292.5693367 2360.4610921
```

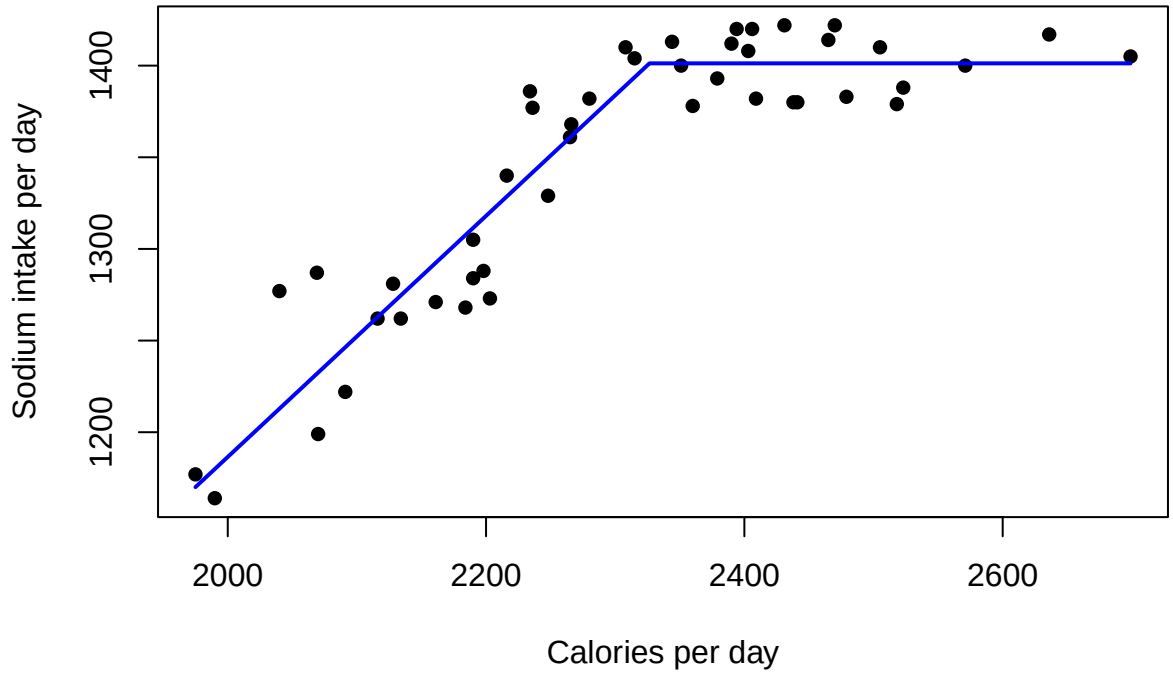
```
set.seed(321)
Boot <- nlsBoot(model)
summary(Boot)
```

```
##
## ------
## Bootstrap statistics
##         Estimate   Std. error
## a   -132.9546386 115.30400068
## b      0.6596725   0.05336634
## clx 2327.5607562  17.69426060
##
## ------
## Median of bootstrap estimates and percentile confidence intervals
##           Median         2.5%        97.5%
```

```
## a   -125.9862894 -368.1354419   86.7305748
## b      0.6570168    0.5574993    0.7691194
## clx 2327.0594447 2293.1666938 2366.0420865
```

Y finalmente dibujar la linea ajustada

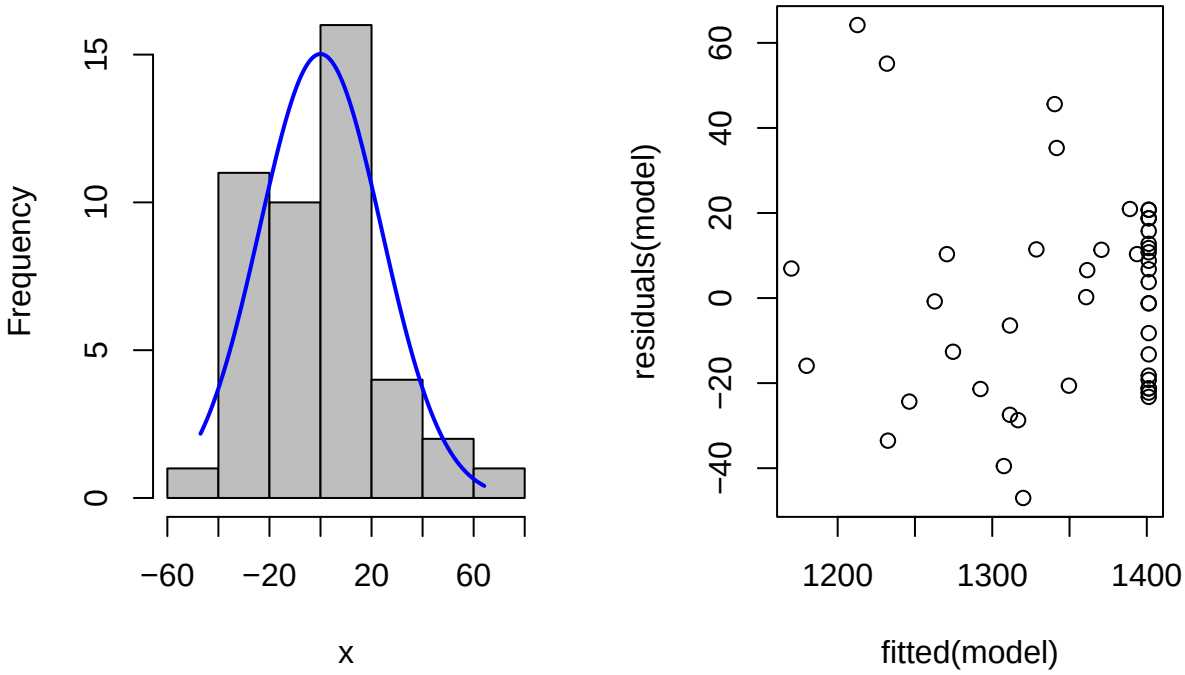
```
# library(rcompanion)
plotPredy(data  = Data,
          x     = Calories,
          y     = Sodium,
          model = model,
          xlab  = "Calories per day",
          ylab  = "Sodium intake per day")
```



El gráfico anterior muestra el consumo diario de sodio vs. el consumo diario de calorías para esos estudiantes. La linea corresponde al ajuste del modelo de regresión con meseta. El valor crítico o de quiebra es 2327 calorías y la meseta es $y = 1401$ mg.. El modelo es significativo $p < 0.0001$ y la pseudo R^2 es 0.892.

Como en cualquier modelo se pueden estudiar los residuos.

```
x <- residuals(model)
par(mfrow=c(1,2))
plotNormalHistogram(x)
plot(fitted(model), residuals(model))
```



En el libro de Mangiafico, el ejemplo continua con el ajuste a un modelo cuadrático con meseta.

Regresión segmentada

Los casos que hemos visto hasta ahora son todos de regresión segmentada en dos trozos. Vamos a utilizar el paquete `segmented` que permite más de un punto de quiebra.

En primer lugar, utilizaremos las funciones de este paquete en los casos que ya hemos resuelto antes con otros procedimientos.

Con los datos del geyser Old Faithful del parque de Yellowstone tenemos:

```
library(segmented)
my.seg <- segmented(lmod0, seg.Z = ~ eruptions, psi = list(eruptions = 3.25))
summary(my.seg)
```

```
##
##  ***Regression Model with Segmented Relationship(s)***
##
## Call:
## segmented.lm(obj = lmod0, seg.Z = ~eruptions, psi = list(eruptions = 3.25))
##
## Estimated Break-Point(s):
##                  Est. St.Err
## psi1.eruptions 3.767  0.208
##
## Coefficients of the linear terms:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   28.1404     2.0009   14.06   <2e-16 ***
## eruptions     13.0834     0.8379   15.61   <2e-16 ***
```

```
## U1.eruptions  -8.0945     1.6899   -4.79       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.694 on 268 degrees of freedom
## Multiple R-Squared: 0.8265,  Adjusted R-squared: 0.8246
##
## Boot restarting based on 6 samples. Last fit:
## Convergence attained in 2 iterations (rel. change 1.6332e-07)
```

En este caso partimos del modelo de regresión OLS del conjunto de datos pero sugerimos el punto de corte como hemos hecho anteriormente. El modelo segmentado nos propone un punto de quiebra estimado alternativo que coincide plenamente con el calculado en el apartado de estimación óptima. Los coeficientes estimados lo son respecto al punto de quiebra óptimo. También se comprueba que esta solución óptima es muy parecida a la que se ha obtenido con la función `nlsLM` del paquete `minpack.lm`.

Probamos ahora con los datos del ejemplo de Mangiafico.

```
my.seg <- segmented(fit.lm, seg.Z = ~ Calories, psi = 2300)
summary(my.seg)
```

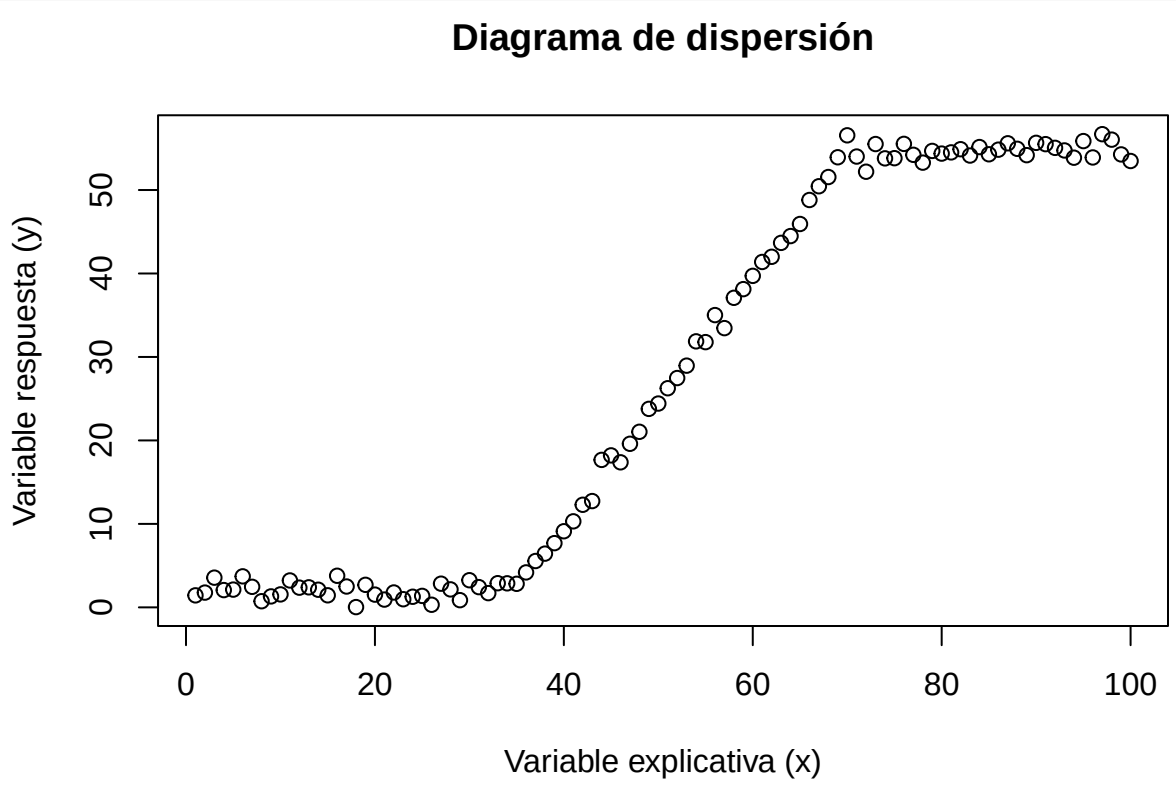
```
##
##  ***Regression Model with Segmented Relationship(s)***
##
## Call:
## segmented.lm(obj = fit.lm, seg.Z = ~Calories, psi = 2300)
##
## Estimated Break-Point(s):
##                    Est. St.Err
## psi1.Calories 2324.925 21.034
##
## Coefficients of the linear terms:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -128.85537  117.49072  -1.097    0.279
## Calories       0.65768    0.05407  12.165 3.46e-15 ***
## U1.Calories   -0.64980    0.08057  -8.065       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 24.76 on 41 degrees of freedom
## Multiple R-Squared: 0.8921,  Adjusted R-squared: 0.8842
##
## Boot restarting based on 6 samples. Last fit:
## Convergence attained in 2 iterations (rel. change 1.7485e-13)
```

Tanto el punto de quiebra, como los coeficientes estimados son similares a los obtenidos con la función `nls`. La poca diferencia entre las pendientes indica que la segunda parte de la regresión es plana.

Por último vamos a simular unos datos con dos puntos de quiebra.

```
set.seed(123)
x <- 1:100
```

```r
y <- 2 + 1.5 * pmax(x - 35, 0) - 1.5 * pmax(x - 70, 0) + rnorm(100)
data <- data.frame(x, y)
plot(
  data$x, data$y,
  main = "Diagrama de dispersión",
  xlab = "Variable explicativa (x)",
  ylab = "Variable respuesta (y)")
```




```r
model <- lm(y ~ x, data = data)
seg_model <- segmented(model, seg.Z = ~x, npsi = 2)
summary(seg_model)
```

```
##
##  ***Regression Model with Segmented Relationship(s)***
##
## Call:
## segmented.lm(obj = model, seg.Z = ~x, npsi = 2)
##
## Estimated Break-Point(s):
##           Est. St.Err
## psi1.x 34.822  0.297
## psi2.x 69.903  0.309
##
## Coefficients of the linear terms:
##              Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)  2.135446   0.324313   6.585 2.59e-09 ***
## x           -0.006914   0.016165  -0.428     0.67
## U1.x         1.501906   0.022380  67.110       NA
## U2.x        -1.471734   0.024173 -60.882       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9247 on 94 degrees of freedom
## Multiple R-Squared: 0.9985,  Adjusted R-squared: 0.9984
##
## Boot restarting based on 6 samples. Last fit:
## Convergence attained in 2 iterations (rel. change 3.4201e-12)
```

Si no indicamos que hay dos puntos de quiebra solo encuenta uno. Mejor si le decimos que hay dos puntos con el parámetro `npsi`.

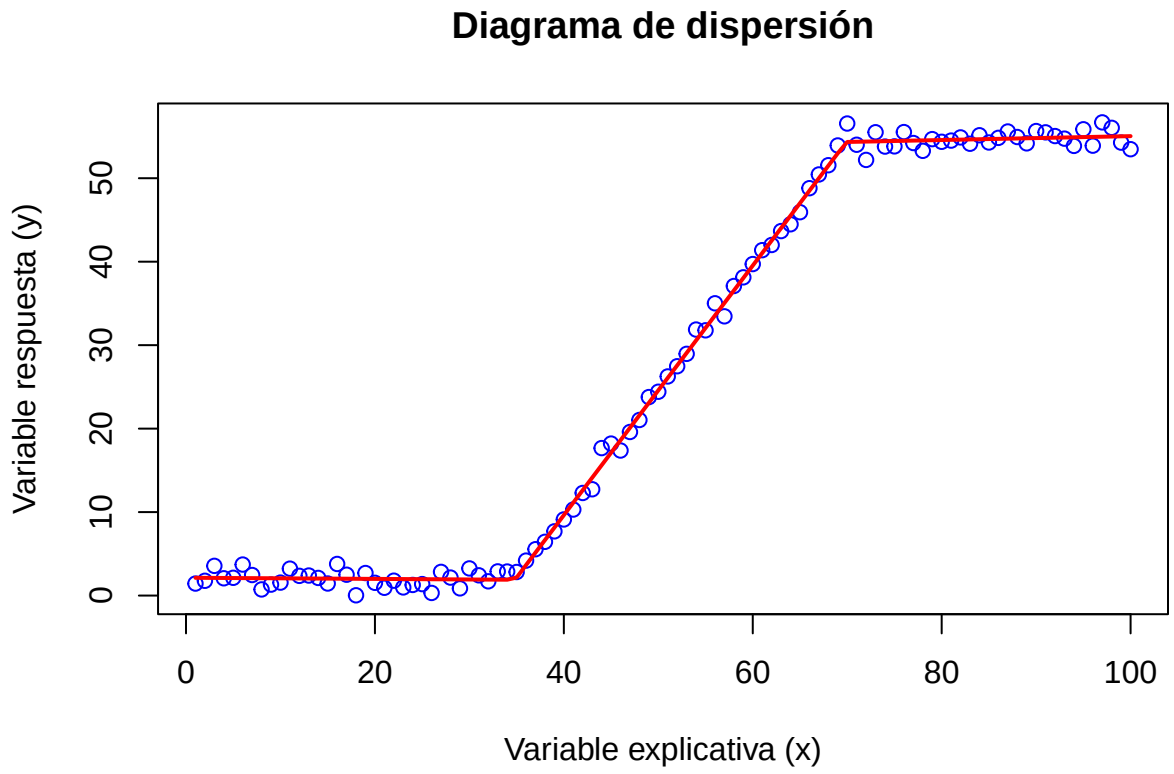
Con las funciones del paquete `segmented` se pueden obtener, entre otras cosas, intervalos de confianza para los puntos de corte.

```
confint(seg_model)
```

```
##           Est. CI(95%).low CI(95%).up
## psi1.x 34.8224     34.2332    35.4117
## psi2.x 69.9030     69.2886    70.5174
```

El gráfico resumen es

```
seg_preds <- predict(seg_model)
plot(
  data$x, data$y,
  main = "Diagrama de dispersión",
  xlab = "Variable explicativa (x)",
  ylab = "Variable respuesta (y)",
  col = "blue"
)
lines(data$x, seg_preds,col = "red", lwd = 2)
```



Referencias

The Chow test in R: A case study of Yellowstone's Old Faithful Geyser

Faraway, J.J. (2014), Linear Models with R, Second Edition, CRC Press.

Mangiafico, Salvatore S., Summary and Analysis of Extension Program Evaluation in R

Piecewise linear regression model (Curso de la PennState)

www.stat.sc.edu/~hitchcock/raw_piecewise_Rexample705.txt

Apéndice

Datos del ejemplo del modelo de regresión con meseta de Mangiafico.

```
Input = ("
Instructor       Grade   Weight  Calories Sodium  Score
'Brendon Small'     6      43     2069    1287      77
'Brendon Small'     6      41     1990    1164      76
'Brendon Small'     6      40     1975    1177      76
'Brendon Small'     6      44     2116    1262      84
'Brendon Small'     6      45     2161    1271      86
'Brendon Small'     6      44     2091    1222      87
'Brendon Small'     6      48     2236    1377      90
'Brendon Small'     6      47     2198    1288      78
'Brendon Small'     6      46     2190    1284      89
'Jason Penopolis'   7      45     2134    1262      76
'Jason Penopolis'   7      45     2128    1281      80
'Jason Penopolis'   7      46     2190    1305      84
'Jason Penopolis'   7      43     2070    1199      68
'Jason Penopolis'   7      48     2266    1368      85
'Jason Penopolis'   7      47     2216    1340      76
'Jason Penopolis'   7      47     2203    1273      69
'Jason Penopolis'   7      43     2040    1277      86
'Jason Penopolis'   7      48     2248    1329      81
'Melissa Robins'    8      48     2265    1361      67
'Melissa Robins'    8      46     2184    1268      68
'Melissa Robins'    8      53     2441    1380      66
'Melissa Robins'    8      48     2234    1386      65
'Melissa Robins'    8      52     2403    1408      70
'Melissa Robins'    8      53     2438    1380      83
'Melissa Robins'    8      52     2360    1378      74
'Melissa Robins'    8      51     2344    1413      65
'Melissa Robins'    8      51     2351    1400      68
'Paula Small'       9      52     2390    1412      78
'Paula Small'       9      54     2470    1422      62
'Paula Small'       9      49     2280    1382      61
'Paula Small'       9      50     2308    1410      72
'Paula Small'       9      55     2505    1410      80
'Paula Small'       9      52     2409    1382      60
'Paula Small'       9      53     2431    1422      70
'Paula Small'       9      56     2523    1388      79
'Paula Small'       9      50     2315    1404      71
'Coach McGuirk'    10      52     2406    1420      68
'Coach McGuirk'    10      58     2699    1405      65
'Coach McGuirk'    10      57     2571    1400      64
'Coach McGuirk'    10      52     2394    1420      69
'Coach McGuirk'    10      55     2518    1379      70
'Coach McGuirk'    10      52     2379    1393      61
'Coach McGuirk'    10      59     2636    1417      70
'Coach McGuirk'    10      54     2465    1414      59
```

```
'Coach McGuirk'    10      54     2479    1383      61
")
Data <- read.table(textConnection(Input), header=TRUE)
###  Order factors by the order in data frame
###  Otherwise, R will alphabetize them
Data$Instructor <- factor(Data$Instructor,
                          levels=unique(Data$Instructor))
### Remove unnecessary objects
rm(Input)
```