Mínimos cuadrados no lineales

Víctor Morales Oñate
Sitio personal
ResearchGate
GitHub
LinkedIn

25 de abril de 2019

Contents

Paquetes de esta sección

```
if(!require(deSolve)){install.packages("deSolve")}
```

Todos los modelos que hemos analizado hasta ahora han sido lineales en los parámetros (es decir, lineales en los beta). El modelo de regresión final fue solo una combinación lineal de predictores de orden superior.

Ahora estamos interesados en estudiar el **modelo de regresión no lineal**:

$$Y = f(\mathbf{X}, \beta) + \epsilon$$

donde $\mathbf{X}$ es un vector de p predictores, β es un vector de k parámetros, $f(\cdot)$ es una función de regresión conocida y ϵ es un término de error cuya distribución puede o no ser normal.

Observe que ya no necesariamente tenemos la dimensión del vector de parámetros simplemente uno mayor que el número de predictores. Algunos ejemplos de modelos de regresión no lineal son:

$$y_i = \frac{e^{\beta_0 + \beta_1 x_i}}{1 + e^{\beta_0 + \beta_1 x_i}} + \epsilon_i$$

$$y_i = \frac{\beta_0 + \beta_1 x_i}{1 + \beta_2 e^{\beta_3 x_i}} + \epsilon_i$$

$$y_i = \beta_0 + (0.4 - \beta_0) e^{-\beta_1 (x_i - 5)} + \epsilon_i$$

Sin embargo, hay algunos modelos no lineales que en realidad se llaman intrínsecamente lineales porque se pueden hacer lineales en los parámetros mediante una simple transformación. Por ejemplo:

$$Y = \frac{\beta_0 X}{\beta_1 + X}$$

puede ser re escrito como:

$$\frac{1}{Y} = \frac{1}{\beta_0} + \frac{\beta_1}{\beta_0} \frac{1}{X}$$

$$\frac{1}{Y} = \theta_0 + \theta_1 \frac{1}{X}$$

Lo cual es lineal en los parámetros transformados θ y θ. En tales casos, la transformación de un modelo a su forma lineal a menudo proporciona mejores procedimientos de inferencia e intervalos de confianza, pero uno debe ser consciente de los efectos que la transformación tiene en la distribución de los errores.

Mínimos cuadrados no lineales

Volviendo a los casos en los que no es posible transformar el modelo a una forma lineal, considere la configuración

$$Y_i = f(\mathbf{X}_i, \beta) + \epsilon_i,$$

donde ϵ es iid normal con media 0 y varianza constante. Para esta configuración, podemos confiar en parte de la teoría de los mínimos cuadrados. Para otros términos de error no normales, se deben emplear diferentes técnicas.

Primero, sea

$$Q = \sum_{i=1}^{n} (y_i - f(\mathbf{X}_i, \beta))^2.$$

Para hallar:

$$\hat{\beta} = \arg\min_{\beta} Q,$$

Primero encontramos cada una de las derivadas parciales de Q con respecto a β_j. Luego, establecemos cada una de las derivadas parciales igual a 0 y los parámetros β_k son reemplazados por $\hat{\beta}_k$ Las funciones a resolver no son lineales en los parámetros $\hat{\beta}_k$ estimados y, a menudo, son difíciles de resolver, incluso en los casos más simples. Por lo tanto, a menudo se emplean métodos numéricos iterativos. ¡Aún más dificultades surgen en que pueden ser posibles múltiples soluciones!

Los algoritmos para la estimación de mínimos cuadrados no lineales incluyen:

- El **método de Newton**, un método clásico basado en un enfoque de gradiente pero que puede ser computacionalmente desafiante y muy dependiente de *buenos* valores iniciales.
- El **algoritmo de Gauss-Newton**, una modificación del método de Newton que proporciona una buena aproximación de la solución a la que debería haber llegado el método de Newton pero que no está garantizado que converja.
- El **método de Levenberg-Marquardt**, que puede resolver las dificultades de cálculo que surgen con los otros métodos, pero puede requerir una tediosa búsqueda del valor óptimo de un parámetro de ajuste.

La función `nls`

Sirve para encontrar parámetros en funciones no lineales. Veamos.

Ajuste de una curva

Se desea ajustar:

$$y(\theta) = x^{\theta} + \epsilon_i$$

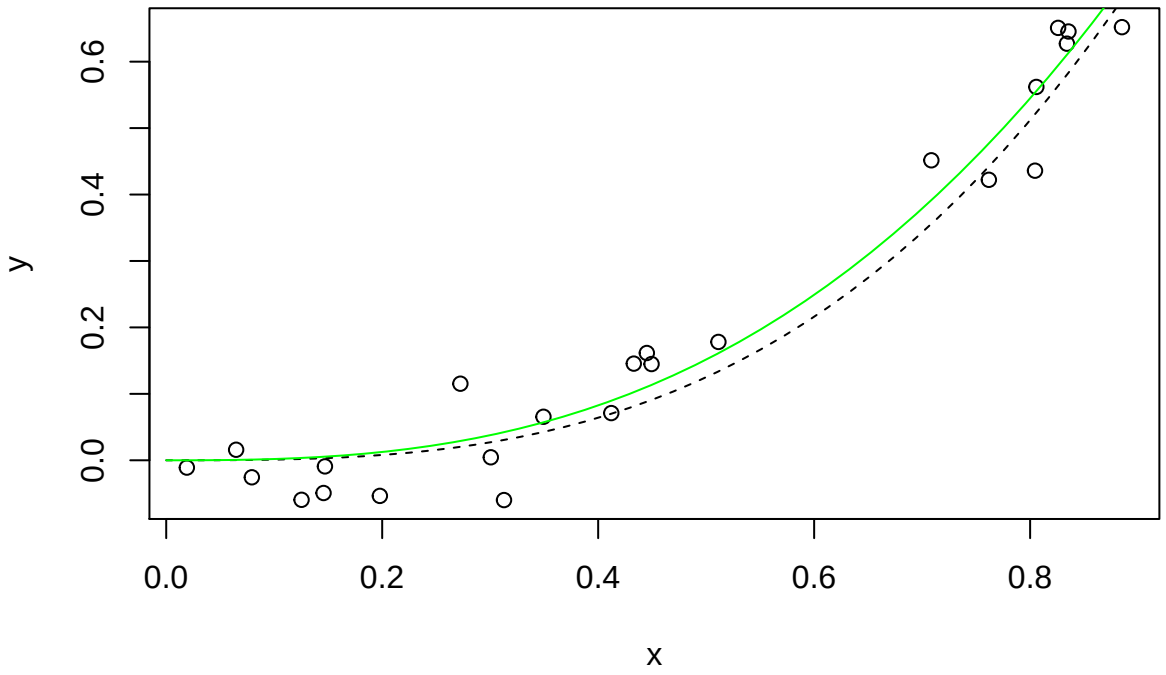
donde ϵ_i sigue una distribución uniforme.

```r
len <- 24
x = runif(len)
y = x^3 + runif(len, min = -0.1, max = 0.1)
plot(x, y)
s <- seq(from = 0, to = 1, length = 50)
lines(s, s^3, lty = 2)

df <- data.frame(x, y)
m <- nls(y ~ I(x^power), data = df, start = list(power = 1), trace = T)
```

```
## 1.529678 :  1
## 0.2514682 :  1.863976
## 0.07299071 :  2.534335
## 0.06688788 :  2.713237
## 0.06687612 :  2.721736
## 0.06687612 :  2.721806
```

```r
lines(s, predict(m, list(x = s)), col = "green")
```



Veamos los resultados:

```r
summary(m)
```

```
##
## Formula: y ~ I(x^power)
##
## Parameters:
##       Estimate Std. Error t value Pr(>|t|)
## power   2.7218     0.1345   20.23 3.76e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.05392 on 23 degrees of freedom
##
```

```
## Number of iterations to convergence: 5
## Achieved convergence tolerance: 7.03e-07
```

Calculando el R cuadrado

```
(RSS.p <- sum(residuals(m)^2))  # Suma de cuadrados de los residuos
```

```
## [1] 0.06687612
```

```
(TSS <- sum((y - mean(y))^2))  # Suma total de residuos
```

```
## [1] 1.606688
```

```
(1 - (RSS.p/TSS))  # R-squared measure
```

```
## [1] 0.9583764
```

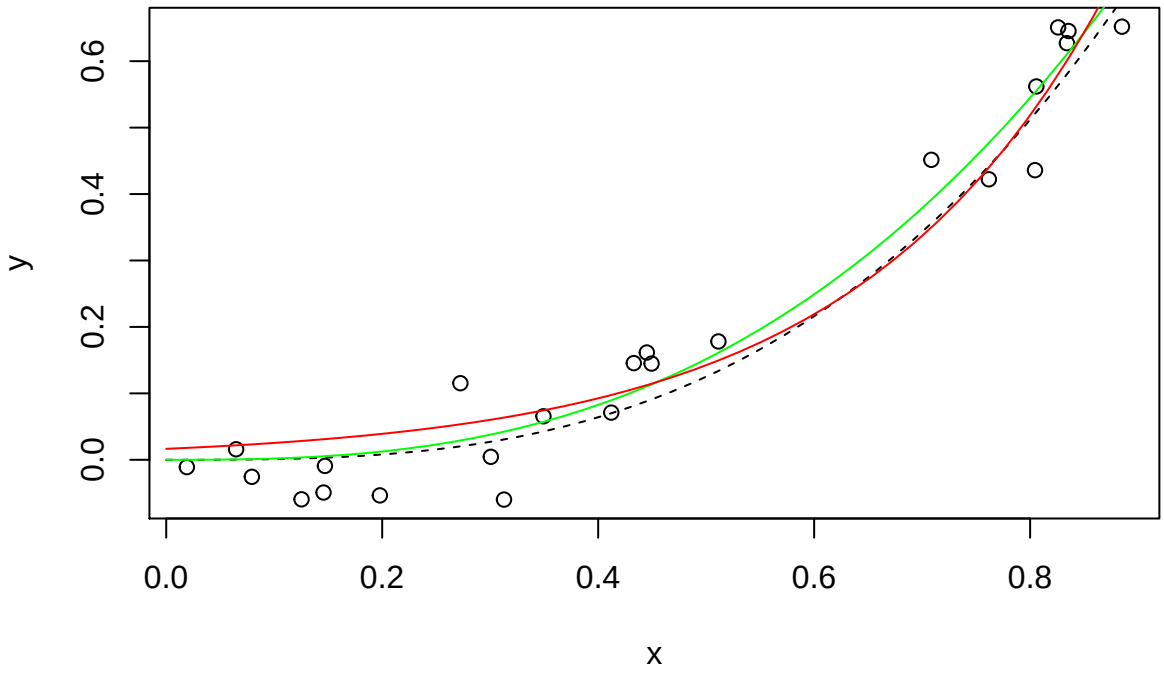
Ajuste a una función exponencial

Redefinamos el modelo m de modo que sea exponencial y volvamos a calcular los resultados:

```
m.exp <- nls(y ~ I(a*exp(b*x)), data = df, start = list(a = 0.5, b = 0.5), trace = T)
```

```
## 4.952587 :  0.5 0.5
## 0.9144959 :  0.1868669 1.1016775
## 0.7009145 :  0.05330896 2.11321322
## 0.5026472 :  0.03347772 2.84132519
## 0.3732 :  0.0154744 3.8990723
## 0.117827 :  0.01415801 4.60687445
## 0.0937941 :  0.01716484 4.24183782
## 0.09308783 :  0.01624847 4.32675604
## 0.09307572 :  0.01657192 4.30387678
## 0.09307505 :  0.01649179 4.31006929
## 0.09307501 :  0.01651398 4.30842623
## 0.09307501 :  0.01650813 4.30886375
## 0.09307501 :  0.01650969 4.30874735
## 0.09307501 :  0.01650928 4.30877834
```

```
plot(x, y)
lines(s, s^3, lty = 2)
lines(s, predict(m, list(x = s)), col = "green")
lines(s, predict(m.exp, list(x = s)), col = "red")
```



De lo anterior, podemos ver que una función exponencial, con la forma $y = \alpha e^{\beta x}$ también se ajusta bien a los datos, aunque la curva parece demasiado pronunciada hacia el lado derecho de la gráfica .

Para probar si este modelo se ajusta mejor a los datos, podemos volver a ejecutar el R cuadrado:

```
(RSS.p <- sum(residuals(m.exp)^2))  # Suma de cuadrados de los residuos
```

```
## [1] 0.09307501
```

```
(TSS <- sum((y - mean(y))^2))  # Suma total de residuos
```

```
## [1] 1.606688
```

```
1 - (RSS.p/TSS)  # R-squared
```

```
## [1] 0.9420703
```

Está claro que la curva exponencial se ajusta menos a los datos.

Ajustando datos a una función predefinida

Podemos usar la función `nls` para alterar de manera iterativa cualquier número de coeficientes en cualquier función que nos interese usar.

Asumamos brevemente que la relación y y x no es en realidad cúbica sino exponencial, con la siguiente forma: $y = e^{a+bsin(x)}$. La función se puede definir utilizando el método genérico.

```
exp.eq <- function(x, a, b) {
    exp(1)^(a + b * sin(x^4))
}
exp.eq(2, 1, 3)  # Hacemos una prueba de la ecuación
```

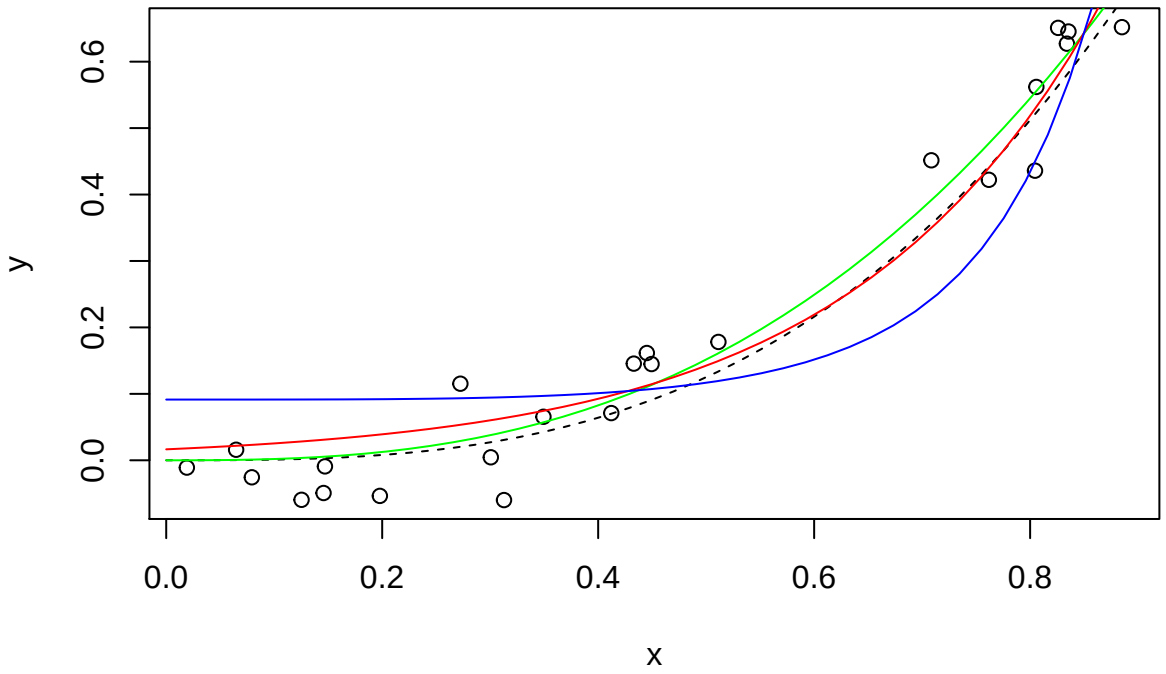
```
## [1] 1.146014
```

Ahora que se ha definido esta forma funcional, `nls` se puede usar para encontrar los coeficientes que mejor reflejan los datos:

```
m.sinexp <- nls(y ~ exp.eq(x, a, b), data = df, start = list(a = 1, b = 1),
    trace = T)
```

```
## 222.7734 :  1 1
## 27.24179 :  0.01372203 1.27049921
## 3.023955 :  -0.9450739  1.8873610
## 0.4552671 :  -1.809618  2.955758
## 0.2848563 :  -2.360297  3.892018
## 0.2832757 :  -2.394630  3.913013
## 0.2832739 :  -2.391544  3.905424
## 0.2832736 :  -2.392874  3.908384
## 0.2832736 :  -2.392356  3.907232
## 0.2832736 :  -2.392558  3.907681
## 0.2832736 :  -2.392480  3.907506
## 0.2832736 :  -2.392510  3.907574
## 0.2832736 :  -2.392498  3.907548
```

```r
plot(x, y)
lines(s, s^3, lty = 2)
lines(s, predict(m, list(x = s)), col = "green")
lines(s, predict(m.exp, list(x = s)), col = "red")
lines(s, predict(m.sinexp, list(x = s)), col = "blue")
```



Claramente, la función exponencial final era tonta y no se ajustaba bien a los datos. Sin embargo, nos mostró cómo podemos usar la función `nls` para ajustar nuestras funciones predefinidas a los datos existentes.

El Problema de los valores iniciales

Encontrar buenos valores iniciales es muy importante en la regresión no lineal para permitir que el algoritmo del modelo converja. Si se establece valores de parámetros iniciales completamente fuera del rango de los valores de los parámetros potenciales, el algoritmo fallará o devolverá parámetros no sensitivos como, por ejemplo, devolver una tasa de crecimiento de 1000 cuando el valor real es 1.04.

La mejor manera de encontrar el valor inicial correcto es hacer una *inspecci´pon gráfica* de los datos, representándolos y basándose en el entendimiento que tiene de la ecuación para encontrar valores de inicio aproximados para los parámetros.

Ejemplo: crecimiento poblacional

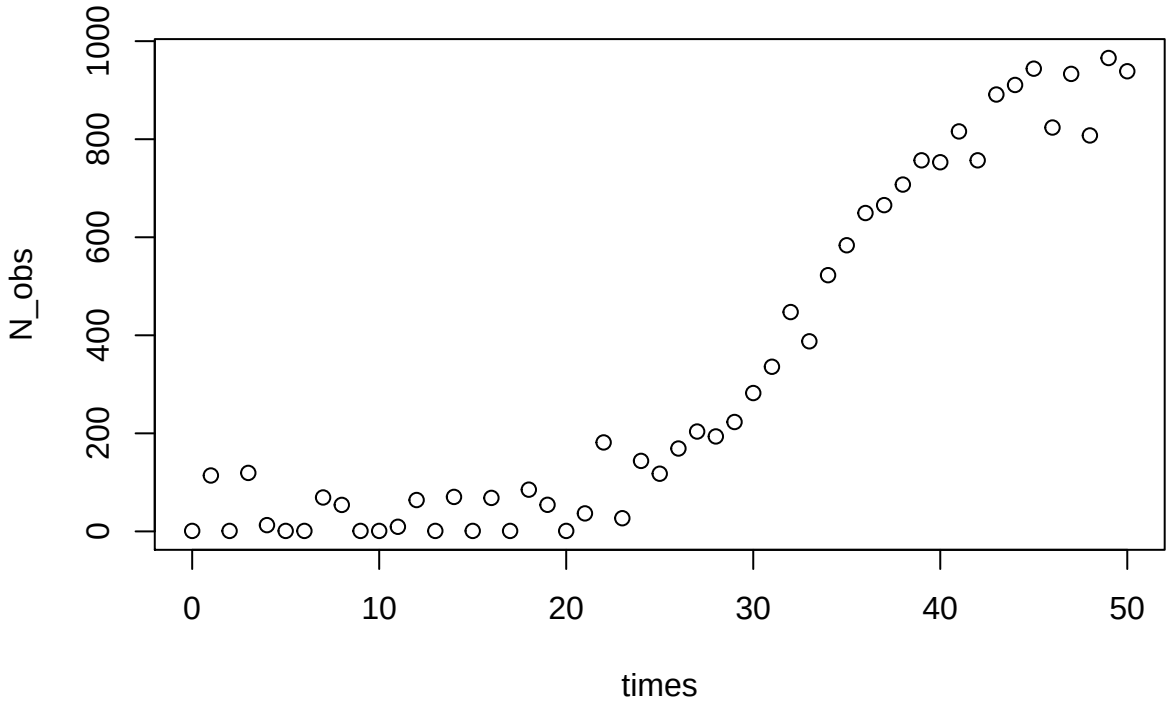
Es muy común que diferentes campos científicos utilicen diferentes parametrizaciones (es decir, diferentes ecuaciones) para el mismo modelo, un ejemplo es el modelo de crecimiento logístico de la población, en ecología se usa la siguiente forma:

$$N_t = \frac{K * N_0 * e^{r*t}}{K + N_0 * (e^{r*t} \lor 1)}$$

Siendo N_t el número de individuos en el momento t, r siendo la tasa de crecimiento de la población y K la capacidad de carga. Podemos reescribir esto como una ecuación diferencial:

$$dN/dt = R * N * (1 - N/K)$$

```
library(deSolve)
#simulamos el crecimiento exponencial con la ecuacion logistica y estimamos los parametros usando nls
log_growth <- function(Time, State, Pars) {
  with(as.list(c(State, Pars)), {
    dN <- R*N*(1-N/K)
    return(list(c(dN)))
  })
}
#los parametros para la ecuacion logistica
pars  <- c(R=0.2,K=1000)
# los parametros iniciales
N_ini  <- c(N=1)
#los pasos de tiempo para evaluar la funcion ode
times <- seq(0, 50, by = 1)
#la ecuaciones diferenciales ordinarias (ODE)
out   <- ode(N_ini, times, log_growth, pars)
# agregamos variaciones aleatorias
N_obs<-out[,2]+rnorm(51,0,50)
# quitamos los valores mayores que 1
N_obs<-ifelse(N_obs<1,1,N_obs)
#plot
plot(times,N_obs)
```



Esta parte fue solo para simular algunos datos con un error aleatorio, ahora viene la parte difícil para estimar los valores iniciales.

Ahora R tiene una función incorporada para estimar los valores iniciales para el parámetro de una ecuación logística (`SSlogis`) pero usa la siguiente ecuación:

$$N_t = \frac{\alpha}{1 + e^{\frac{xmid - t}{scale}}}$$

```
#encontramos los parametros de la ecuacion
SS<-getInitial(N_obs~SSlogis(times,alpha,xmid,scale),data=data.frame(N_obs=N_obs,times=times))
```

Usamos la función `getInitial` que proporciona algunas conjeturas iniciales sobre los valores de los parámetros en función de los datos. Pasamos a esta función un modelo de inicio automático (`SSlogis`) que toma como argumento un vector de entrada (los valores de t donde se evaluará la función), y los nombres de los tres parámetros para la ecuación logística.

Sin embargo, como los `SSlogis` utilizan una parametrización diferente, necesitamos usar un poco de álgebra para pasar de los valores estimados de inicio automático devueltos desde `SSlogis` a los que están en la ecuación que queremos usar.

```
#usamos una parametrizacion diferente
K_start<-SS["alpha"]
R_start<-1/SS["scale"]
N0_start<-SS["alpha"]/(exp(SS["xmid"]/SS["scale"])+1)
#la formula del modelo
log_formula<-formula(N_obs~K*N0*exp(R*times)/(K+N0*(exp(R*times)-1)))
#ajustamos el modelo
m<-nls(log_formula,start=list(K=K_start,R=R_start,N0=N0_start))
#parametros estimados
summary(m)
```

```
##
## Formula: N_obs ~ K * N0 * exp(R * times)/(K + N0 * (exp(R * times) - 1))
```

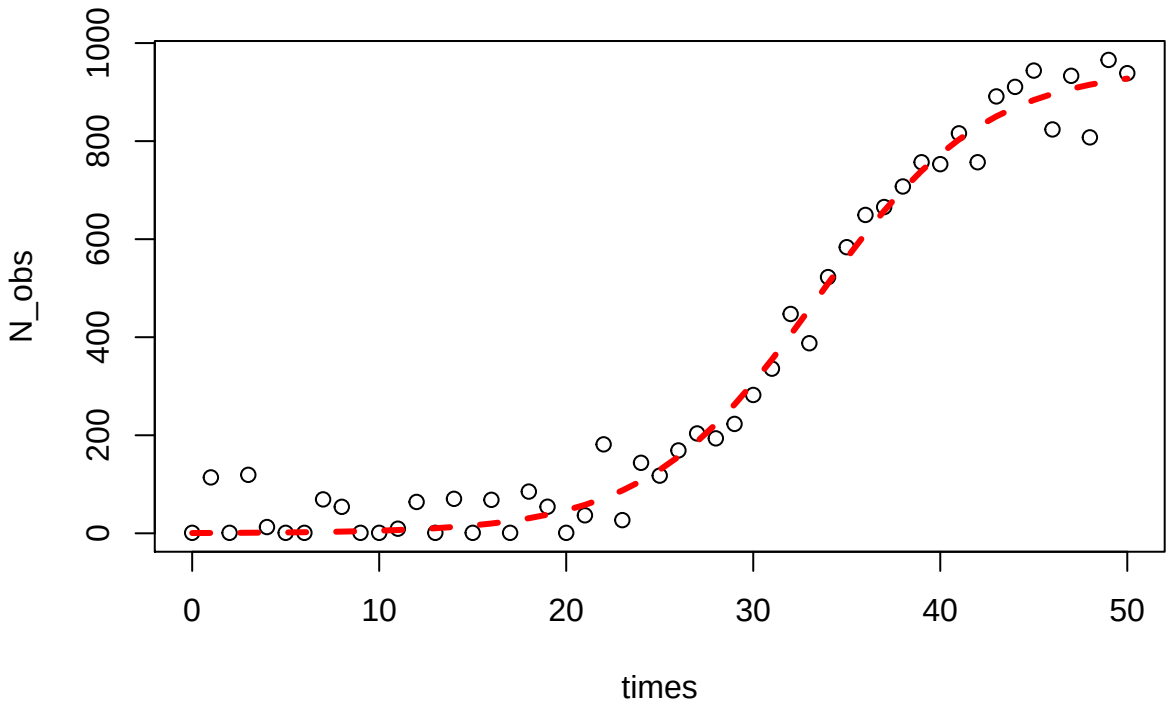
```
##
## Parameters:
##           Estimate Std. Error t value Pr(>|t|)
## K.alpha  950.78139   26.91860  35.321   <2e-16 ***
## R.scale    0.22143    0.01655  13.378   <2e-16 ***
## N0.alpha   0.59027    0.30561   1.931   0.0593 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 48.29 on 48 degrees of freedom
##
## Number of iterations to convergence: 0
## Achieved convergence tolerance: 3.523e-06
```

```
#bondad de ajuste
cor(N_obs,predict(m))
```

```
## [1] 0.9915543
```

Graficamos

```
plot(times,N_obs)
lines(times,predict(m),col="red",lty=2,lwd=3)
```



Ejemplo: Cobb-Douglas

El modelo:

$$Y_i = \beta_1 X_{2i}^{\beta_2} X_{3i}^{\beta_3} e^{u_i}$$

donde

- Y: producción

- X_2: insumo trabajo
- X_3: insumo capital
- u: término de perturbación
- e: base del logaritmo

Notemos que el modelo es multiplicativo, si tomamos la derivada obetenemos un modelo más familiar respecto a la regresión lineal múltiple:

$$lnY_i = ln\beta_1 + \beta_2 ln(X_{2i}) + \beta_3 ln(X_{3i}) + u_i$$

```
uu <- "https://raw.githubusercontent.com/vmoprojs/DataLectures/master/tabla7_3.csv"
# datos <- read.csv(file="tabla7_3.csv",sep=";",dec=".",header=TRUE)
datos <- read.csv(url(uu),sep=";",dec=".",header=TRUE)
```

```
W <- log(datos$X2)

K <- log(datos$X3)

LY <- log(datos$Y)

reg.1 <- lm(LY~W+K)
summary(reg.1)
```

```
##
## Call:
## lm(formula = LY ~ W + K)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.15919 -0.02917  0.01179  0.04087  0.09640
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -3.3387     2.4491  -1.363 0.197845
## W             1.4987     0.5397   2.777 0.016750 *
## K             0.4899     0.1020   4.801 0.000432 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0748 on 12 degrees of freedom
## Multiple R-squared:  0.8891, Adjusted R-squared:  0.8706
## F-statistic: 48.08 on 2 and 12 DF,  p-value: 1.864e-06
```

Si se desea evaluar la hipótesis $\beta_2 + \beta_3 = 1$, se usa (`vcov(fit.model)`):

$$t = \frac{(\hat{\beta}_2 + \hat{\beta}_3) - 1}{\sqrt{var(\hat{\beta}_2) + var(\hat{\beta}_3) + 2cov(\hat{\beta}_2, \hat{\beta}_3)}}$$

Si se desea estimar el modelo de forma no lineal, tenemos

```
produccion <- function(a,b1,b2,X2,X3)
{
  y <- a * (X2^(b1)) * (X3^(b2))
  y
```

```
}

# nls(produccion(a,b1,b2,X2,X3), data=datos, start =
#       c(a=exp(-3.6529493),b1=1.0376775,b2 = 0.7187662), trace = TRUE)

cd.nls <- nls(Y~a * (X2^(b1)) * (X3^(b2)), data=datos, start =
      c(a=exp(-3.3),b1=1.1,b2 = 0.4), trace = TRUE)
```

```
## 8708132824 :  0.03688317 1.10000000 0.40000000
## 8675849643 :  0.02100187 1.18968838 0.40929049
## 8587365262 :  0.01442634 1.26102994 0.41656110
## 8419670086 :  0.008348122 1.368361221 0.427335004
## 7881005064 :  0.005458484 1.494727501 0.439617864
## 6583036265 :  0.0059799 1.5761620 0.4469403
## 4808546360 :  0.008585682 1.604184859 0.449075421
## 2619382635 :  0.01335838 1.61312719 0.44961401
## 663361319 :  0.0205383 1.6148192 0.4496652
## 28677021 :  0.0277159 1.6147623 0.4496533
## 28676896 :  0.02771594 1.61477622 0.44965674
```

```
summary(cd.nls)
```

```
##
## Formula: Y ~ a * (X2^(b1)) * (X3^(b2))
##
## Parameters:
##    Estimate Std. Error t value Pr(>|t|)
## a   0.02772    0.05775   0.480  0.63990
## b1  1.61478    0.42800   3.773  0.00266 **
## b2  0.44966    0.07430   6.052 5.74e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1546 on 12 degrees of freedom
##
## Number of iterations to convergence: 10
## Achieved convergence tolerance: 3.735e-07
```

Taller

Recuerde que el error en los mínimos cuadrados ordinarios es aditivo. Si no cuenta con esa estructura, busque la forma función que lo sea.

Considere la siguiente ecuación:

$$Y = \frac{\epsilon}{1 + e^{\beta_1 X_1 + \beta_2 X_2}}$$

$$log(Y) = -log(1 + e^{\beta_1 X_1 + \beta_2 X_2}) + log(\epsilon)$$

La segunda ecuación es la versión transformada que usaremos para la estimación.

- Usando como semilla =1, simule 100 valores de X_1 y X_2 con distribución uniforme entre 0 y 1.
- Usando el modelo propuesto, simule valores de Y donde $a = 0.8$ y $b = 0.5$.

- El ruido tiene distribución normal con media cero y varianza 1.

Debe obteener los siguientes resultados:

```
##
## Formula: log(Y) ~ -log(1 + exp(a * X1 + b * X2))
##
## Parameters:
##   Estimate Std. Error t value Pr(>|t|)
## a   0.8470     0.2697   3.141  0.00223 **
## b   0.5962     0.2718   2.193  0.03065 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6517 on 98 degrees of freedom
##
## Number of iterations to convergence: 3
## Achieved convergence tolerance: 2.299e-06
```