Data 609 - Module4

Amit Kapoor

10/08/2021

Contents

Ex. 1

For Example 19 on Page 79 in the book, carry out the regression using R.

x	-0.98	1.00	2.02	3.03	4.00
y	2.44	-1.51	-0.47	2.54	7.52

Solution

```
x <- c(-0.98, 1.00, 2.02, 3.03, 4.00)
y <- c(2.44, -1.51, -0.47, 2.54, 7.52)

model1 <- lm(y~x)
summary(model1)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##       1       2       3       4       5
##  2.9547 -2.8511 -2.7671 -0.7037  3.3671
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.4038     2.2634   0.178    0.870
## x             0.9373     0.9058   1.035    0.377
##
## Residual standard error: 3.481 on 3 degrees of freedom
## Multiple R-squared:  0.263,  Adjusted R-squared:  0.01739
## F-statistic: 1.071 on 1 and 3 DF,  p-value: 0.3769
```

So the equations is y = 0.9373 x + 0.4038

Ex. 2

Implement the non-linear curve-fitting of Example 20 on Page 83 for the following data:

x	0.10	0.50	1.00	1.50	2.00	2.50
y	0.10	0.28	0.40	0.40	0.37	0.32

Solution

We have R defined as

$R = y_i - \frac{x_i}{a+bx_i{}^2}$

```r
# Given values
x <- c(0.10, 0.50, 1.00, 1.50, 2.00, 2.50)
y <- c(0.10, 0.28, 0.40, 0.40, 0.37,0.32)

# given function
R <- function(x,y,a,b){
  y - x/(a+b*x^2)
}

# partial derivative of R w.r.t a and b
pd.Ra <- function(x,a,b){
  x/(a+b*x^2)^2
}
pd.Rb <- function(x,a,b){
  x^3/(a+b*x^2)^2
}

# find a and b
find_params <- function(x,y,a,b,iter_params){

  residuals <- matrix(0,1,length(x))
  for (i in 1:length(x)) {
    rid <- R(x[i], y[i],a,b)
    residuals[1,i] <- rid
  }

  # jacobian matrix
  jacob.mat <- matrix(0,length(x),2)

  for (i in 1:length(x)){
    jacob.mat[i,1] <- pd.Ra(x[i],a,b)
    jacob.mat[i,2] <- pd.Rb(x[i],a,b)
  }
  # initial jacobian matrix
  jacob.mat

  params <- iter_params - solve(t(jacob.mat) %*% jacob.mat) %*% t(jacob.mat) %*% t(residuals)
  params
}
```

Now the first iteration using the Gauss-Newton Algo gives a and b as

```r
a <- 1
b <- 1
iter_params <- matrix(1,2,1)
p1 <- find_params(x,y,a,b,iter_params)
p1
```

```
##          [,1]
## [1,] 1.344879
## [2,] 1.031709
```

After second iteration using the Gauss-Newton Algo gives a and b as

```r
a <- p1[1,1]
b <- p1[2,1]

p2 <- find_params(x,y,a,b,p1)
p2
```

```
##          [,1]
## [1,] 1.474156
## [2,] 1.005853
```

After third iteration using the Gauss-Newton Algo gives a and b as

```r
a <- p2[1,1]
b <- p2[2,1]

p3 <- find_params(x,y,a,b,p2)
p3
```

```
##          [,1]
## [1,] 1.485228
## [2,] 1.002223
```

We can see it converges to a = ~1.49 and b = ~1

Next we will try to solve it with R.

```r
x <- c(0.10, 0.50, 1.00, 1.50, 2.00, 2.50)
y <- c(0.10, 0.28, 0.40, 0.40, 0.37, 0.32)

df1 <- data.frame(x, y)
eq1 <- function(x, a, b){
  x / (a + b * x^2)
}

# model Nonlinear Least Squares
model2 <- nls(y ~ eq1(x, a, b), data = df1, start = list(a = 1, b = 1))
model2
```

```
## Nonlinear regression model
##   model: y ~ eq1(x, a, b)
##    data: df1
##     a     b
## 1.485 1.002
##  residual sum-of-squares: 0.00121
##
## Number of iterations to convergence: 5
```

```
## Achieved convergence tolerance: 3.899e-07
```

We can see the a and b values come up as 1.485 and 1 which is similar to above followed manual steps.

Ex. 3

For the data with binary y values, try to fit the following data

x	0.1	0.5	1.0	1.5	2.0	2.5
y	0	0	1	1	1	0

to the nonlinear function

$y = \frac{1}{1+e^{a+bx}}$,

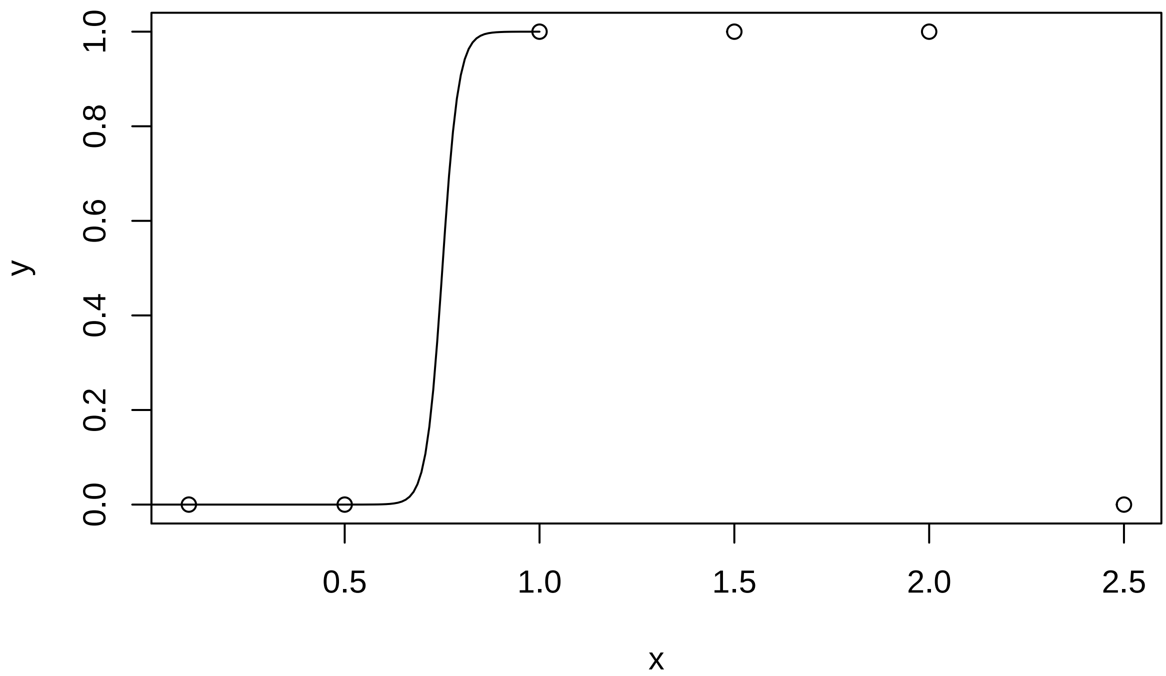
starting with a = 1 and b = 1.

Solution

```
# given values
x <- c(0.1, 0.5, 1.0, 1.5, 2.0, 2.5)
y <- c(0, 0, 1, 1, 1, 0)

df2 <- data.frame(x, y)

# given function
eq2 <- function(x, a, b){
  1 / (1+exp(a + b*x))
}

# model Nonlinear Least Squares
model3 <- nls(y ~ eq2(x, a, b), data = df2, start = list(a = 1, b = 1))
model3
```

```
## Nonlinear regression model
##   model: y ~ eq2(x, a, b)
##    data: df2
##      a      b
##  36.42 -48.51
##  residual sum-of-squares: 1
##
## Number of iterations to convergence: 12
## Achieved convergence tolerance: 7.656e-06
```

```
s <- seq(from = 0, to = 1, length = 100)
# plot
plot(x, y)
lines(s, predict(model3, list(x = s)))
```



1.0
0.8
0.6
0.4
0.2
0.0
y
0.5
1.0
1.5
2.0
2.5
x