HW4 - DATA 609

Thomas Hill

October 17, 2021

```
library(knitr)
```

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

To solve this problem, the can use the general formula from equation 4.57 in the textbook.

```
ex1_matrix <- matrix(c(5, sum(x), sum(x^2), sum(x), sum(x^2), sum(x^3), sum(x^2), sum(x^3), sum
(x^4)), nrow = 3)

ex1_y_matrix <- matrix(c(sum(y), sum(y*x), sum(y * x^2)))




ex1_sol <- solve(ex1_matrix) %*% ex1_y_matrix

print(ex1_sol)
```

```
##            [,1]
## [1,] -0.5055154
## [2,] -2.0261594
## [3,]  1.0065065
```

Using this method, a0 = -0.506, a1 = -2.026, and a2 = 1.007, which is approximately equal to the answer in the textbook.

$$\hat{y} = -0.506 - 2.026x + 1.007x^2$$

Ex. 2 – Implement the nonlinear curve-fitting of Example 20 on Page 83 for the following data:

x	0.1	0.50	1.0	1.5	2.00	2.50
y	0.1	0.28	0.4	0.4	0.37	0.32

The curve-fitting model of example 20 attempts to fit a regression of the format

$$\hat{y} = \frac{x}{a + bx^2}$$

by minimizing the sum of residual squares. This is done by the Gauss-Newton algorithm, which in the logistic case solves for a 2 x 1 vector of values for a and b. Specifically, this is accomplished in Example 20 by obtaining the Jacobian matrix of the data and recursively calculating a1 given the following:

$$a_{t+1} = a_t + (J^T J)^{-1} J^T R_{a_t}$$

Where J is the Jacobian, R is the residual matrix of the current guess, and a_0 are estimates of the parameters.

The Jacobian is calculated by finding partial derivatives of R with respect to a and b, which equal

$$\frac{\partial R}{\partial a} = \frac{x_i}{(a + bx_i^2)^2}, \frac{\partial R}{\partial b} = \frac{x_i^3}{(a + bx_i^2)^2}$$

and then forming a 2 x 6 matrix with $\frac{\partial R}{\partial a}$ in one column and $\frac{\partial R}{\partial b}$ in the next.

```
a_0 <- 1
b_0 <- 1

initial_params <- matrix(a_0,b_0, nrow = 2)

drda <- x/(initial_params[1] + initial_params[2]*x^2)^2

drdb <- (x^3)/(initial_params[1] + initial_params[2]*x^2)^2


ex2_j <- cbind(drda,drdb)

print(ex2_j)
```

```
##            drda        drdb
## [1,] 0.09802960 0.000980296
## [2,] 0.32000000 0.080000000
## [3,] 0.25000000 0.250000000
## [4,] 0.14201183 0.319526627
## [5,] 0.08000000 0.320000000
## [6,] 0.04756243 0.297265161
```

```
ex2_resid <- matrix(c(y - x/(a_0 + b_0 * x^2)))

print(ex2_resid)
```

```
##              [,1]
## [1,]  0.000990099
## [2,] -0.120000000
## [3,] -0.100000000
## [4,] -0.061538462
## [5,] -0.030000000
## [6,] -0.024827586
```

```
initial_input <- list(initial_params, ex2_resid, ex2_j)

ex2_solve <- function(list_input = initial_input) {
  params <- list_input[[1]]
  R <- list_input[[2]]
  J <- list_input[[3]]
  ex2_solve.resid <- R

  newton <- params - solve(t(J) %*% J) %*% t(J) %*% ex2_solve.resid #gauss-newton algorithm

  ex2_solve.result <- newton #t + 1 iteration of parameter estimates
  ex2_solve.resid <- y - x/(newton[1] + newton[2] * x^2) #recalculate residual matrix

  ex2_drda <- x/(ex2_solve.result[1] + ex2_solve.result[2]*x^2)^2

  ex2_drdb <- (x^3)/(ex2_solve.result[1] + ex2_solve.result[2]*x^2)^2

  ex2_solve.jacobian <- cbind(ex2_drda,ex2_drdb) #recalculate jacobian

    return(list(ex2_solve.result,ex2_solve.resid, ex2_solve.jacobian))
  }

first_iteration <- ex2_solve()
print(first_iteration)
```

```
## [[1]]
##          [,1]
## drda 1.344879
## drdb 1.031709
##
## [[2]]
## [1]  0.0262099473 -0.0319528492 -0.0207713033 -0.0091403110  0.0044838426
## [6] -0.0007982838
##
## [[3]]
##        ex2_drda     ex2_drdb
## [1,] 0.05444972 0.0005444972
## [2,] 0.19462916 0.0486572901
## [3,] 0.17704849 0.1770484897
## [4,] 0.11159720 0.2510936911
## [5,] 0.06680103 0.2672041227
## [6,] 0.04116462 0.2572788472
```

The initial iteration gives approximately the same parameters as the solution in the book. Lets see how further iterations do.

```
ex2_solve(ex2_solve())
```

```
## [[1]]
##          [,1]
## drda 1.474156
## drdb 1.005853
##
## [[2]]
## [1]  0.032624310 -0.009750988 -0.003224242 -0.001356452  0.006202872
## [6] -0.002134254
##
## [[3]]
##        ex2_drda     ex2_drdb
## [1,] 0.04539484 0.0004539484
## [2,] 0.16791127 0.0419778175
## [3,] 0.16258979 0.1625897890
## [4,] 0.10739133 0.2416305022
## [5,] 0.06617418 0.2646967014
## [6,] 0.04150819 0.2594261935
```

During the second iteration, the answers remain the same. Lets also check whether RSS is still decreasing during this process:

```
print(sum(ex2_resid^2))
```

```
## [1] 0.02970437
```

```
print(sum(first_iteration[[2]]^2))
```

```
## [1] 0.00224368
```

```
print(sum(ex2_solve(first_iteration)[[2]]^2))
```

```
## [1] 0.001214694
```

It appears that RSS is still decreasing during my implementation of the Gauss-Newton method.

Ex. 3 – For the data with binary *y* values, try to fit the following data

x	0.1	0.5	1	1.5	2	2.5
y	0.0	0.0	1	1.0	1	0.0

to the nonlinear function

$$y = \frac{1}{1 + e^{\alpha + \beta x}}$$

starting with a = 1 and b = 1.

Using the Gauss-Newton method from Exercise 2, we can estimate the parameters. In this case, the partial derivatives that make up the Jacobian are:

$$\frac{\partial R}{\partial a} = \frac{e^{a+bx}}{(1+e^{a+bx})^2}, \frac{\partial R}{\partial b} = \frac{xe^{a+bx}}{(1+e^{a+bx})^2}$$

```
a_0 <- 1
b_0 <- 1

initial_params <- matrix(a_0,b_0, nrow = 2)

drda <- exp(initial_params[1] + initial_params[2]*x)/(1 + exp(initial_params[1] + initial_params
[2]*x))^2

drdb <- x *exp(initial_params[1] + initial_params[2]*x)/(1 + exp(initial_params[1] + initial_par
ams[2]*x))^2



ex3_j <- cbind(drda,drdb) #initial jacobian

print(ex3_j)
```

```
##            drda       drdb
## [1,] 0.18736988 0.01873699
## [2,] 0.14914645 0.07457323
## [3,] 0.10499359 0.10499359
## [4,] 0.07010372 0.10515557
## [5,] 0.04517666 0.09035332
## [6,] 0.02845302 0.07113256
```

```
ex3_resid <- matrix(c(y - x/(1+exp(a_0 + b_0 * x)))) #initial residuals

print(ex3_resid)
```

```
##             [,1]
## [1,] -0.02497399
## [2,] -0.09121276
## [3,]  0.88079708
## [4,]  0.88621273
## [5,]  0.90514825
## [6,] -0.07328058
```

```
initial_input <- list(initial_params, ex3_resid, ex3_j)

ex3_solve <- function(list_input = initial_input) {
  params <- list_input[[1]]
  R <- list_input[[2]]
  J <- list_input[[3]]
  ex3_solve.resid <- R

  newton <- params - solve(t(J) %*% J) %*% t(J) %*% R #gauss-newton algorithm

  ex3_solve.result <- newton #t + 1 iteration of parameter estimates
  ex3_solve.resid <- y - 1/(1+exp(ex3_solve.result[1] + ex3_solve.result[2] * x)) #recalculate r
esidual matrix

  drda <- exp(ex3_solve.result[1] + ex3_solve.result[2]*x)/(1 + exp(ex3_solve.result[1] + ex3_so
lve.result[2]*x))^2

  drdb <- x *exp(ex3_solve.result[1] + ex3_solve.result[2]*x)/(1 + exp(ex3_solve.result[1] + ex3
_solve.result[2]*x))^2

  ex3_solve.jacobian <- cbind(drda,drdb) #recalculate jacobian

    return(list(ex3_solve.result,ex3_solve.resid, ex3_solve.jacobian))
  }

first_iteration <- ex3_solve()
print(first_iteration)
```

```
## [[1]]
##           [,1]
## drda  2.717535
## drdb -6.816731
##
## [[2]]
## [1] -1.154887e-01 -6.661516e-01  1.631540e-02  5.486170e-04  1.816625e-05
## [6] -9.999994e-01
##
## [[3]]
##              drda         drdb
## [1,] 1.021511e-01 1.021511e-02
## [2,] 2.223937e-01 1.111968e-01
## [3,] 1.604921e-02 1.604921e-02
## [4,] 5.483161e-04 8.224741e-04
## [5,] 1.816592e-05 3.633185e-05
## [6,] 6.012269e-07 1.503067e-06
```

```
ex3_solve(ex3_solve())
```

```
## [[1]]
##           [,1]
## drda  3.629436
## drdb -2.789558
##
## [[2]]
## [1] -0.03387943 -0.09668113  0.69843960  0.36472817  0.12458829 -0.96592291
##
## [[3]]
##            drda        drdb
## [1,] 0.03273161 0.003273161
## [2,] 0.08733389 0.043666944
## [3,] 0.21062173 0.210621726
## [4,] 0.23170153 0.347552297
## [5,] 0.10906605 0.218132095
## [6,] 0.03291584 0.082289605
```

```
ex3_solve(ex3_solve(ex3_solve()))
```

```
## [[1]]
##            [,1]
## drda -1.1434447
## drdb -0.4228437
##
## [[2]]
## [1] -0.7659763 -0.7949241  0.1727462  0.1445861  0.1203487 -0.9002992
##
## [[3]]
##            drda       drdb
## [1,] 0.17925659 0.01792566
## [2,] 0.16301977 0.08150988
## [3,] 0.14290492 0.14290492
## [4,] 0.12368096 0.18552144
## [5,] 0.10586490 0.21172980
## [6,] 0.08976052 0.22440129
```

```
ex3_solve(ex3_solve(ex3_solve(ex3_solve())))
```

```
## [[1]]
##           [,1]
## drda  2.261722
## drdb -1.347829
##
## [[2]]
## [1] -0.1065060 -0.1696926  0.7137961  0.5597083  0.3931868 -0.7517281
##
## [[3]]
##            drda        drdb
## [1,] 0.09516244 0.009516244
## [2,] 0.14089704 0.070448519
## [3,] 0.20429123 0.204291230
## [4,] 0.24643492 0.369652373
## [5,] 0.23859095 0.477181898
## [6,] 0.18663294 0.466582349
```

After looking at a few iterations, it appears the parameters are not approaching a single value. Lets also look at the residuals:

```
print(sum(ex3_resid^2))
```

```
## [1] 2.394783
```

```
print(sum(ex3_solve()[[2]]^2))
```

```
## [1] 1.457361
```

```
print(sum(ex3_solve(ex3_solve())[[2]]^2))
```

```
## [1] 1.579869
```

```
print(sum(ex3_solve(ex3_solve(ex3_solve()))[[2]]^2))
```

```
## [1] 2.094393
```

```
print(sum(ex3_solve(ex3_solve(ex3_solve(ex3_solve())))[[2]]^2))
```

```
## [1] 1.582608
```

It appears that the first iteration provides the lowest RSS. Next, lets look at what values are given using R’s built-in regression function

```
ex3_data <- data.frame(cbind(x,y))

ex3_lm <- glm(data = ex3_data, formula = y~., family = binomial(link = 'logit'))

ex3_lm$coefficients
```

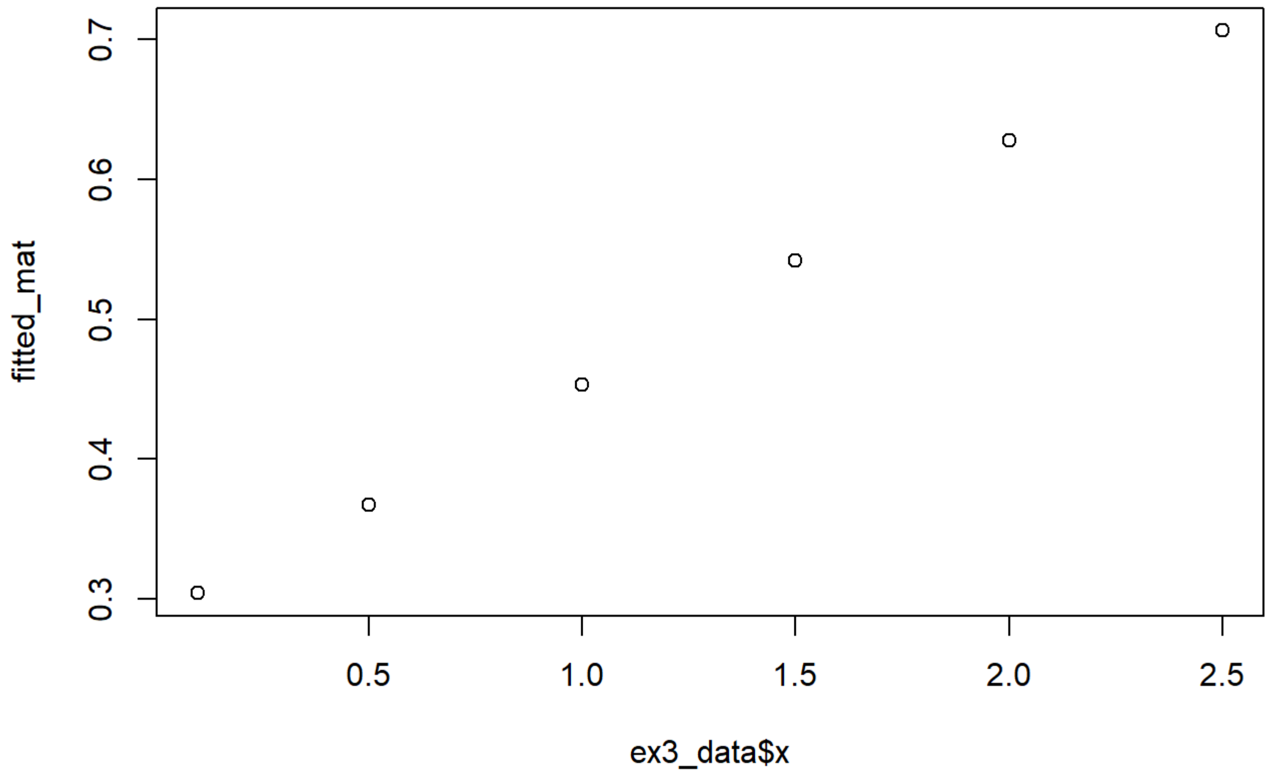
```
## (Intercept)           x
##  -0.8982069   0.7099480
```

```
fitted_mat <- ex3_lm$fitted.values

sum((y-fitted_mat)^2)
```

```
## [1] 1.374187
```

```
plot(fitted_mat ~ ex3_data$x)
```



Using the built-in function, an RSS of 1.374 is found using parameters a = -0.898 and b = 0.7099. This is lower than performance from the Gauss-Newton method.