**Andrew Hillard**

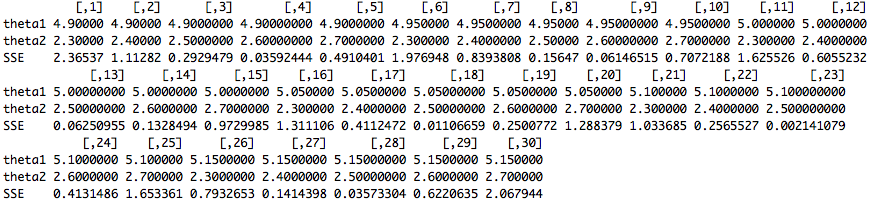
**Data Analysis 2**

**Homework 1**

Problem 1

1. I created a function in R that calculated the SSE for each combination of the candidate values for and .

The SSE estimates are given below. Out of the candidate values, the best estimate, or candidate values that produced the lowest SSE, was when and , giving a SSE = 0.0021410. See column 23 of the matrix below. The next best estimate was when and , giving a SSE = 0.01106659.



1. Assuming multiplicative errors, the natural log of the model is given by the following equation.

We can treat the above equation as a linear model where plays the role of , plays the role of plays the role of , and plays the role of . Fitting this linear model using R, we get the following estimates for and .



We then solve for and to get the reasonable starting values of and

Problem 2

I found my starting values as follows.

1. I took the equation and set x to zero. When x=0, . I then plugged in the data point (0, 2.5). (0, 2.5) is the approximate value of the y-intercept based on the shape of the provided data. I rewrote the equation as .
2. I took the equation and set y to zero. I then plugged in (2.5, 0), which was my estimate of x when y equals zero based on the provided data. I solved for to get .
3. I rewrote the original equation in terms of . I then took the data point (1.9, 0.5), rounded to (2, 0.5), and plugged it into the equation in terms of . I solved for and proceeded to solve for the other parameters.

This gave me the following starting values: 2.5, -1, 0.

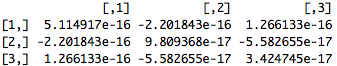
The Gauss Newton Algorithm gave me the following parameter estimates.

4.0

-1.5

0.6

The asymptotic variance covariance matrix is given below.



The estimated is 2.100351e-16.

The parameter estimates converged at the 11th Iteration.

The objective function, or SSE, at the convergence values was 5.187867e-14.

The R Code is given in the Appendix.

I encountered two interesting problems. First, I ran the algorithm as a Gauss-Newton without the modification of step halving. The result was that the estimated parameter values never converged. Second, I ran the Modified Gauss Newton with bad starting values due to a calculation error early on. The estimated values after convergence were not correct. This goes to show you the importance of starting values.

Appendix

data <- read.table('S16hw1pr2.dat', col.names=c("x","y"))

thetaNames <- c("t0", "t1", "t2")

startingValues <- c(2.5, -1, 0)

equation <- expression((t0+t1\*x)/(1+t2\*exp(0.4\*x)))

xValues <- data[,1]

yValues <- data[,2]

F <- function(thetaNames, thetaValues, xValues, f){

Fmatrix <- matrix(nrow = length(xValues), ncol = length(thetaValues))

for (i in 1 : length(xValues)){

for (j in 1 : length(thetaValues)){

df.dtheta <- D(f, thetaNames[j])

Fmatrix[i,j] <- eval(df.dtheta, list(x=xValues[i], t0=thetaValues[1], t1=thetaValues[2], t2=thetaValues[3]))

}

}

Fmatrix

}

f <- function(thetaValues, xValues, f) {

fmatrix <- matrix(nrow = length(xValues), ncol=1)

for (i in 1 : length(xValues)){

fmatrix[i,1] <- eval(equation, list(x=xValues[i], t0 = thetaValues[1], t1=thetaValues[2], t2=thetaValues[3]))

}

fmatrix

}

SSE <- function(xValues, yValues, tValues, equation) {

SSE <- 0

for (i in 1: length(xValues)) {

SSE <- SSE + (yValues[i]-eval(equation, list(x=xValues[i], t0=tValues[1], t1=tValues[2], t2=tValues[3])))^2

}

SSE

}

gaussNewton <- function(thetaNames, startingValues, xValues, yValues, equation, iterations) {

gaussNewtonMatrix <- matrix(nrow=(1+length(startingValues)), ncol=iterations)

currentThetaValues <- startingValues

for ( i in 1: iterations) {

Fmatrix <- F(thetaNames, currentThetaValues, xValues, equation)

fequation <- f(currentThetaValues, xValues, equation)

FFinverse <- solve(t(Fmatrix) %\*% Fmatrix)

delta <- FFinverse %\*% t(Fmatrix) %\*% (yValues-fequation)

currentSSE <- SSE(xValues, yValues, currentThetaValues, equation)

potentialSSE <- currentSSE + 1

k <- -1

b <- 0

while (potentialSSE > currentSSE && b < 10) {

k <- k + 1

potentialThetaValues <- currentThetaValues + t(delta)/(2^k)

potentialSSE <- SSE(xValues, yValues, potentialThetaValues, equation)

b <- b + 1

}

currentThetaValues <- currentThetaValues + t(delta)/(2^k)

gaussNewtonMatrix[1,i] <- i

gaussNewtonMatrix[2,i] <- currentThetaValues[1]

gaussNewtonMatrix[3,i] <- currentThetaValues[2]

gaussNewtonMatrix[4,i] <- currentThetaValues[3]

}

gaussNewtonMatrix

}

thetaSolution <- gaussNewton(thetaNames, startingValues, xValues, yValues, equation, 50)[2:4,50]

ObjectiveFunctionValue <- SSE(xValues, yValues, thetaSolution, equation)

EstimatedVariance <- ObjectiveFunctionValue / (250-3)

FSolution <- F(thetaNames, thetaSolution, xValues, equation)

FFInverseSolution <- solve(t(FSolution) %\*% FSolution)

EstimatedVariance \* FFInverseSolution