PA2 - HW1

Kristiyan Dimitrov

Jan 25, 2020

Problem 2

```
'data.frame':    18 obs. of  2 variables:
 $ y: num  2.1 2.5 4.9 5.5 7 8.4 9.6 10.2 11.4 12.5 ...
 $ x: num  1 1.5 2 3 4 5 6 7.5 8.5 10 ...
```

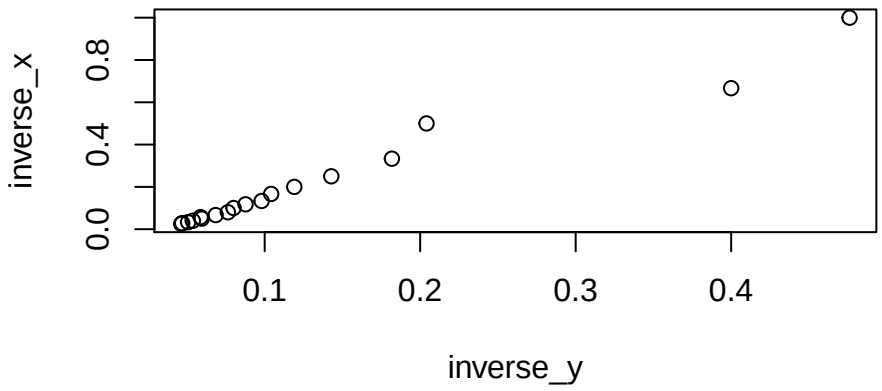
a)

```
[1] "Creating inverse predictor variables"
```

```
    y   x inverse_y inverse_x
1 2.1 1.0 0.4761905 1.0000000
2 2.5 1.5 0.4000000 0.6666667
3 4.9 2.0 0.2040816 0.5000000
4 5.5 3.0 0.1818182 0.3333333
5 7.0 4.0 0.1428571 0.2500000
6 8.4 5.0 0.1190476 0.2000000
```

```
[1] "Fitting Linear Model"
```

```
Call:
lm(formula = inverse_y ~ inverse_x, data = enzyme)

Residuals:
      Min        1Q    Median        3Q       Max
-0.056684 -0.004123  0.000694  0.002766  0.063565

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 0.033759   0.006684   5.051 0.000118 ***
inverse_x   0.454014   0.020061  22.632 1.41e-13 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.02175 on 16 degrees of freedom
Multiple R-squared:  0.9697, Adjusted R-squared:  0.9678
F-statistic: 512.2 on 1 and 16 DF,  p-value: 1.411e-13
```




```
[1] "These are the beta values from linear regression"
```

```
[1] 0.03375868
```

```
[1] 0.454014
```

```
[1] "These are our initial guesses for the gamma parameters based on them"
```

```
[1] 29.62201 13.44881
```

b)

Using nlm()

```
[1] "Encoding non-linear model as function"
```

```
function(gamma){
  yhat = gamma[1]*x / (gamma[2]+x)
  sum((y-yhat)^2) # Return SSE
}
```

```
[1] "Applying nlm() to model"
```

```
$minimum
[1] 4.302271

$estimate
[1] 28.13688 12.57428

$gradient
[1] -1.768662e-07  1.565261e-07
```

```
$hessian
          [,1]      [,2]
[1,]  8.268182 -7.388299
[2,] -7.388299  7.487009

$code
[1] 2

$iterations
[1] 10

[1] "These are the estimates for the Gamma parameters, which give the lowest SSE"

[1] "i.e. these are the MLE estimates"

[1] 28.13688 12.57428
```

Using nls()

```
[1] "Using nls() to compute best gamma parameters, which minimize SSE"

6.587434 :  29.62201 13.44881
4.303552 :  28.14230 12.59805
4.302271 :  28.13786 12.57534
4.302271 :  28.13708 12.57449
4.302271 :  28.13705 12.57445


Formula: y ~ fn2(x1, p)

Parameters:
   Estimate Std. Error t value Pr(>|t|)
p1  28.1370     0.7280   38.65  < 2e-16 ***
p2  12.5745     0.7631   16.48 1.85e-11 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5185 on 16 degrees of freedom

Number of iterations to convergence: 4
Achieved convergence tolerance: 4.347e-07

[1] "These are the estimates for the gamma parameters"

      p1       p2
28.13705 12.57445
```

Problem 3 - Deriving CI for parameters Analytically

a) Fisher Matrix & Covariance Matrix

```
[1] "To get the Information matrix, we need to first find the MSE"
```

```
[1] 0.2688919

[1] "We also need the Hessian"

          [,1]      [,2]
[1,]  8.268182 -7.388299
[2,] -7.388299  7.487009

[1] "Then the Information Matrix is Hessian/2/MSE"

          [,1]      [,2]
[1,]  15.37455 -13.73842
[2,] -13.73842  13.92197

[1] "The covariance matrix for the gamma parameters is the inverse of the information matrix"

          [,1]      [,2]
[1,] 0.5502797 0.5430248
[2,] 0.5430248 0.6076944

[1] "The standard errors of the parameters are the sqrt() of the diagonal entries in the covariance m

[1] 0.7418084 0.7795476
```

b)

```
[1] "Covariance matrix based on nls() output"

          p1        p2
p1 0.5299535 0.5202828
p2 0.5202828 0.5822505

[1] "The SE for the parameters"

       p1        p2
0.7279790 0.7630534
```

We see that the above results are pretty close to the SE results obtained via nlm()

c) - Calculating confidence intervals

```
[1] "These are the confidence intervals for the two variables"

[1] 26.68294 29.59083

[1] 11.04637 14.10219

[1] "CI based on nls() output"

      2.5 %   97.5 %
p1 26.71024 29.56386
p2 11.07890 14.07001
```

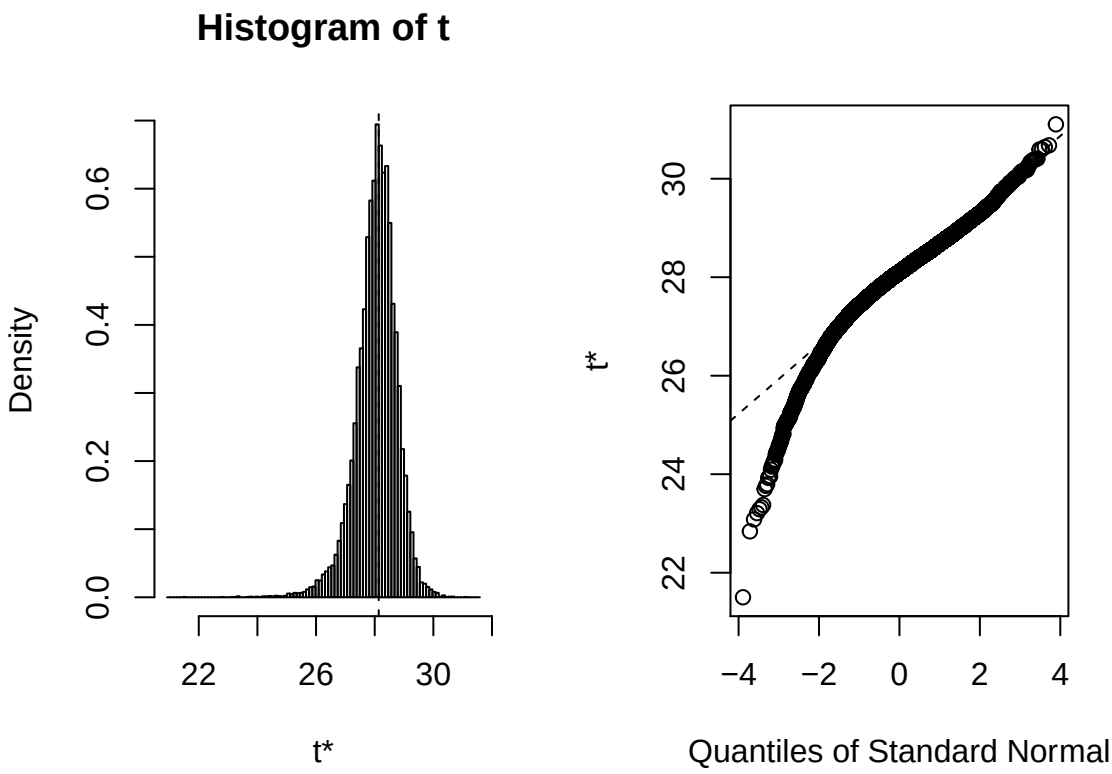
We see that the above nls() results for the gamma CI closely match the results based on nlm()

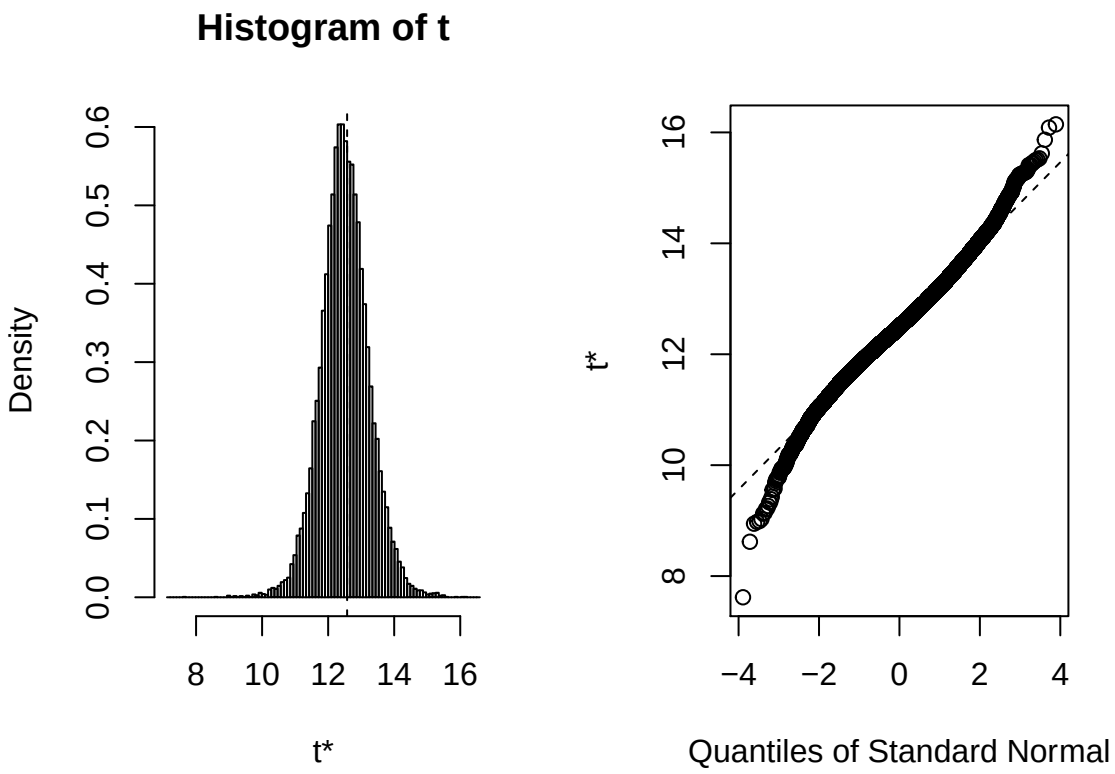
Problem 4 - Using Bootstrapping to construct CIs numerically

```
[1] "Running Bootstrap on non-linear model"


ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = enzyme, statistic = enzymeFit, R = 20000, gammaInitial = gammaStart)


Bootstrap Statistics :
    original      bias    std. error
t1* 28.13688 -0.08958410   0.7050140
t2* 12.57428 -0.06306756   0.7376107

[1] "We see that the estimates from bootstrap match the ones from nlm(), as expected!"

[1] 28.13688 12.57428

[1] 28.13688 12.57428
```

Part a)

```
[1] "Covariance matrix of the bootstrap estimates"

          [,1]      [,2]
[1,] 0.4970448 0.4831124
[2,] 0.4831124 0.5440695

[1] "histogram & Q-Q plot of bootstrap estimates for the two parameters"
```


Histogram of t
Density
0.0
0.2
0.4
0.6
22
26
30
t*
t*
22
24
26
28
30
−4
−2
0
2
4
Quantiles of Standard Normal




```
[1] "the SE for gamma0 & gamma1"

[1] 0.7050140 0.7376107
```

Part b) - Crude CI based on 95% Confidence Intervals from bootstrapped distributions

```
[1] "Crude (Normal) Confidence interval for gamma0 from Bootstrap results"

[1] 26.75506 29.51871

[1] "Crude (Normal) Confidence interval for gamma1 from Bootstrap results"

[1] 11.12856 14.02000

[1] "These are our original estimates. Clearly, they fall within the CIs above"

[1] 28.13688 12.57428
```

Part c) Reflected CI based on Bootstrap

```
[1] "For Gamma0"
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 20000 bootstrap replicates

CALL :
boot.ci(boot.out = enzymeBoot, conf = 0.95, type = "basic", index = 1)

Intervals :
Level      Basic
95%   (27.00, 29.82 )
Calculations and Intervals on Original Scale


[1] "For Gamma1"


BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 20000 bootstrap replicates

CALL :
boot.ci(boot.out = enzymeBoot, conf = 0.95, type = "basic", index = 2)

Intervals :
Level      Basic
95%   (11.14, 14.08 )
Calculations and Intervals on Original Scale
```

Part d)

The Crude Confidence intervals match the Reflected ones pretty closely. There is a bit more of a difference when we look closely at the lower bound of the CIs for gamma0. This is probably due to our bootstrap estimates being a bit more skewed-left than normal distribution. i.e. there is a slight tail to the left. This is supported both by the histogram and the Q-Q plot

Problem 5 - Calculating Prediction Interval for future response Y* at X* = 27

Calculating 95% CI for y_hat at x = 27

Using Bootstrap to estimate prediction

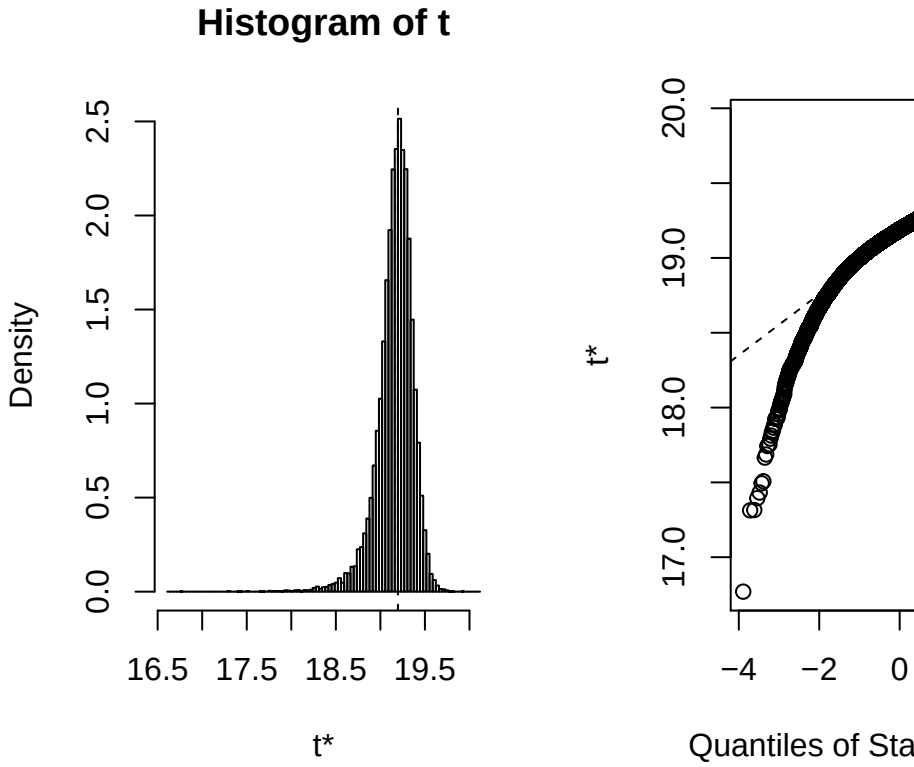
```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = enzyme, statistic = enzymeFit, R = 20000, gammaInitial = gammaStart,
    x_pred = 27)


Bootstrap Statistics :
    original      bias    std. error
t1* 19.19671 -0.03229492   0.2035199
```

The SE for the prediction is

```
[1] 0.2035199
```

Calculating the CI for the predictable part i.e. g(x, gamma)

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 20000 bootstrap replicates

CALL :
boot.ci(boot.out = enzymeBoot, conf = 0.95, type = c("norm",
    "basic"))

Intervals :
Level      Normal              Basic
95%   (18.83, 19.63 )   (18.91, 19.71 )
Calculations and Intervals on Original Scale
```



This is a plot for our predictions i.e. for y_hat

Calculating 95% PI for y_hat at x = 27

```
[1] 18.10489 20.28852
```

I would expect the Prediction interval (the one just above) to contain a future response at X = 27 with 95% certainty. The confidence interval is narrower and corresponds to the expected value of future responses at X = 27. This simply means that if we were to make many many observations at X = 27, then their average would lie in the CI we derived with 95% confidence.

Problem 6

Fitting the linear model with sqrt(x)

```
Call:
lm(formula = y ~ sqrt(x), data = enzyme)

Residuals:
    Min      1Q  Median      3Q     Max
-1.7997 -0.5451  0.2085  0.4413  1.6771

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -0.4566     0.5154  -0.886    0.389
sqrt(x)       3.7720     0.1401  26.918 9.41e-15 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.9483 on 16 degrees of freedom
Multiple R-squared:  0.9784, Adjusted R-squared:  0.977
F-statistic: 724.6 on 1 and 16 DF,  p-value: 9.414e-15
```

Calculating the AIC for the above linear model

```
[1] 2.836148
```

Calculating the AIC for our non-linear model with the logLik function applied to the output of the nls() function

```
'log Lik.' 1.628871 (df=3)
```

We can see that the AIC for our non-linear model is lower than the AIC for the sqrt(x) model/ Let's check if this conclusion will be supported empirically via Cross-Validation

Problem 7 - Cross Validation

Getting the parameter estimates from lm() call as first guess for fitting in CV

```
[1] -0.4565841  3.7720135
```

Running Cross Validation on the linear and non-linear model and calculating MSE for each iteration as well as avg MSE across all iterations

```
[1] "MSE from all 20 CV iterations"

          [,1]     [,2]
 [1,] 0.2943015 1.110655
 [2,] 0.2943015 1.110655
 [3,] 0.2943015 1.110655
 [4,] 0.2943015 1.110655
```

```
 [5,] 0.2943015 1.110655
 [6,] 0.2943015 1.110655
 [7,] 0.2943015 1.110655
 [8,] 0.2943015 1.110655
 [9,] 0.2943015 1.110655
[10,] 0.2943015 1.110655
[11,] 0.2943015 1.110655
[12,] 0.2943015 1.110655
[13,] 0.2943015 1.110655
[14,] 0.2943015 1.110655
[15,] 0.2943015 1.110655
[16,] 0.2943015 1.110655
[17,] 0.2943015 1.110655
[18,] 0.2943015 1.110655
[19,] 0.2943015 1.110655
[20,] 0.2943015 1.110655

[1] "Avg. MSE from all 20 CV iterations"

[1] 0.2943015 1.1106553

[1] "MSE standard deviations"

[1] 0 0

[1] "R^2 for the two models"

[1] 0.9924874 0.9716484
```

Cross Validation shows that the MSE for our non-linear model is .294 while for the linear model its 1.11. Therefore, the non-linear model performs better on new data.

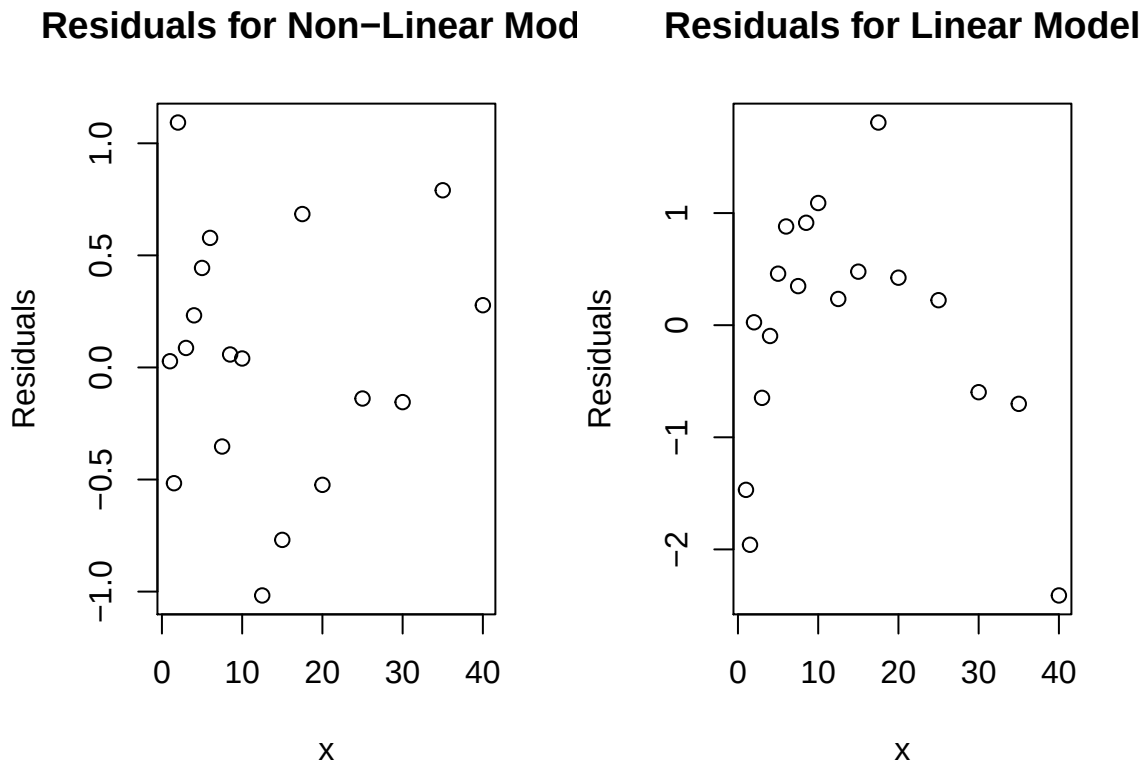
Unfortunately, I wasn't able to find out why all the CV values for MSE are the same. I checked and confirmed that the index randomization for the take-one-out is different every time. I've also printed out yhat and out and out2 to verify that the outcomes of the optimization are different every time.

In the end, I think this might be expected behavior. Every time we create new folds, we are changing the order in which the values are estiamted and added to yhat, but the values end up being the same over all 20 iterations.

Problem 8 - Residual Plots & Comments

```
[1] "Non-linear Residuals"

 [1]  0.02771323 -0.51647761  1.09261534  0.08698839  0.23247110
 [6]  0.44393828  0.57793510 -0.35280044  0.05784193  0.03998987
[11] -1.01717536 -0.76924352  0.68433708 -0.52367597 -0.13860028
[16] -0.15466563  0.79043929  0.27771942

[1] "Linear Residuals"
```

```
 [1] -1.46878982 -1.95871020  0.02559058 -0.64817175 -0.09643711
 [6]  0.45927742  0.88035713  0.34837375  0.91327688  1.08950703
[11]  0.23373806  0.47741743  1.80734041  0.42356015  0.22271085
[16] -0.59788710 -0.70165838 -2.41071732

[1] " This is x"

 [1]  1.0  1.5  2.0  3.0  4.0  5.0  6.0  7.5  8.5 10.0 12.5 15.0 17.5 20.0
[15] 25.0 30.0 35.0 40.0
```



We can certainly see an 'arch' in the residuals for the linear model. In other words, homoscedasticity doesn't seem to be satisfied, which is an important assumption for Non-Linear Least Squares. Therefore, the residual plots also support our conclusion that the non-linear model is more suitable for predicting responses for new observations (particualrly when it comes to large (x>30) or small (x<3) values for x)

Appendix

This section is to be used for including your R code.

```
# define functions used globally which are not dependent on any question specifics
# instead of reloading them each time
# you can also use this space to load R packages
# or custom scripts sourced from a R file
# for example, I have included the CV_ind function
```

```
CVInd <- function(n,K) {
  # n is sample size; K is number of parts;
  # returns K-length list of indices for each part
  m<-floor(n/K) #approximate size of each part
  r<-n-m*K
  I<-sample(n,n) #random reordering of the indices
  Ind<-list() #will be list of indices for all K parts
  length(Ind)<-K
  for (k in 1:K) {
    if (k <= r) kpart <- ((m+1)*(k-1)+1):((m+1)*k)
    else kpart<-((m+1)*r+m*(k-r-1)+1):((m+1)*r+m*(k-r))
    Ind[[k]] <- I[kpart] #indices for kth part of data
  }
  Ind
}
############ QUESTION 2 begins here ###################
enzyme = read.csv('HW1_data.csv', sep=',') # Read in data
enzyme = enzyme[1:2] # Remove empty columns
str(enzyme)
# First create transformed variables
print("Creating inverse predictor variables")
enzyme['inverse_y'] = 1/enzyme$y
enzyme['inverse_x'] = 1/enzyme$x
head(enzyme)
# Now we fit a linear model
print("Fitting Linear Model")
fit = lm(inverse_y ~ inverse_x, data = enzyme)
summary(fit)
plot(enzyme[3:4])
beta0 = fit$coefficients[[1]]
beta1 = fit$coefficients[[2]]
print("These are the beta values from linear regression")
beta0; beta1
# Initial guesses for gamma parameters
gamma0 = 1/beta0
gamma1 = beta1/beta0
gammaStart = c(gamma0, gamma1)
print("These are our initial guesses for the gamma parameters based on them")
gammaStart
y = enzyme[1]
x = enzyme[2]
# Defining our parametric model as a function, which returns SSE
print("Encoding non-linear model as function")
model <- function(gamma){
  yhat = gamma[1]*x / (gamma[2]+x)
  sum((y-yhat)^2) # Return SSE
}
model
print("Applying nlm() to model")
optimizer = nlm(model, p = gammaStart, hessian = TRUE)
optimizer
# These are the estimates for the Gamma parameters, which give the lowest SSE i.e. these are the MLE es
print('These are the estimates for the Gamma parameters, which give the lowest SSE')
```

```r
print('i.e. these are the MLE estimates')
optimizer$estimate
enzyme = read.csv('HW1_data.csv', sep=',') # Read in data
enzyme = enzyme[1:2] # Remove empty columns
x1<-enzyme$x;y<-enzyme$y

# Using nls() to compute best gamma parameters, which minimize SSE
print('Using nls() to compute best gamma parameters, which minimize SSE')
fn2 <- function(x1,p) {p[1]*x1 / (p[2]+x1)}
out2<-nls(y~fn2(x1,p),start=list(p=gammaStart),trace=TRUE)
summary(out2)
print('These are the estimates for the gamma parameters')
out2$m$getPars()
########### QUESTION 3 begins here ###################
# To get the Information matrix, we need to first find the MSE
print('To get the Information matrix, we need to first find the MSE')
MSE = optimizer$minimum/(length(y)-length(optimizer$estimate)) # SSE / df = SSE / (n - p)
MSE
# We also need the Hessian
Hessian = optimizer$hessian # This is the Hessian
print('We also need the Hessian')
Hessian
# Below we find the observer Information matrix
InfoMatrix = Hessian/2/MSE
print('Then the Information Matrix is Hessian/2/MSE')
InfoMatrix
# The covariance matrix for the gamma parameters is the inverse of the information matrix
CovGamma = solve(InfoMatrix)
print('The covariance matrix for the gamma parameters is the inverse of the information matrix')
CovGamma
# The standard errors of the parameters are the sqrt() of the diagonal entries in the covariance matrix
SE = sqrt(diag(CovGamma))
print('The standard errors of the parameters are the sqrt() of the diagonal entries in the covariance ma
SE
print('Covariance matrix based on nls() output')
vcov(out2) # Covariance matrix based on nls() output
print('The SE for the parameters')
sqrt(diag(vcov(out2))) # The SE for the parameters
z = 1.96
# Estimates for gamma0
L.gamma0 = optimizer$estimate[1] - z * SE[1]
U.gamma0 = optimizer$estimate[1] + z * SE[1]
# Estimates for gamma1
L.gamma1 = optimizer$estimate[2] - z * SE[2]
U.gamma1 = optimizer$estimate[2] + z * SE[2]
# These are the confidence intervals for the two variables
print('These are the confidence intervals for the two variables')
c(L.gamma0, U.gamma0)
c(L.gamma1, U.gamma1)
print('CI based on nls() output')
confint.default(out2) # CI based on nls() output
########### QUESTION 4 begins here ###################
# install.packages('boot')
```

```r
set.seed(32)
library(boot)
enzyme = read.csv('HW1_data.csv', sep=',') # Read in data
enzyme = enzyme[1:2] # Remove empty columns
# First we define the function, which produces the statistics we are interested in
# In this case estimates for the gamma parameters
enzymeFit<-function(Z,i,gammaInitial) {
   Zboot<-Z[i,]
   x<-Zboot[[2]];y<-Zboot[[1]]
   model <- function(gamma){
      yhat = gamma[1]*x / (gamma[2]+x)
      sum((y-yhat)^2) # Return SSE
      }
   out<-nlm(model,p=gammaInitial)
   gamma <-out$estimate #parameter estimates
}
# Now we pass that statistics function to the boot command
print('Running Bootstrap on non-linear model')
enzymeBoot = boot(enzyme, enzymeFit, R = 20000, gammaInitial = gammaStart) # gammaStart: results from L
enzymeBoot
# We see that the estimates match, as expected!
print('We see that the estimates from bootstrap match the ones from nlm(), as expected!')
enzymeBoot$t0
optimizer$estimate
enzymeCovMatrix = cov(enzymeBoot$t)
print('Covariance matrix of the bootstrap estimates')
enzymeCovMatrix
print('histogram & Q-Q plot of bootstrap estimates for the two parameters')
par(mfrow=c(1,4))
plot(enzymeBoot, index = 1)
plot(enzymeBoot, index = 2)
# Finding the SE for gamma0 & gamma1
enzymeBootStrapSE = sqrt(diag(enzymeCovMatrix))
print('the SE for gamma0 & gamma1')
enzymeBootStrapSE
z = 1.96
Bootstrap.L.gamma0 = enzymeBoot$t0[1] - z*enzymeBootStrapSE[1]
Bootstrap.L.gamma1 = enzymeBoot$t0[2] - z*enzymeBootStrapSE[2]
Bootstrap.U.gamma0 = enzymeBoot$t0[1] + z*enzymeBootStrapSE[1]
Bootstrap.U.gamma1 = enzymeBoot$t0[2] + z*enzymeBootStrapSE[2]
# Crude (Normal) Confidence interval for gamma0 from Bootstrap results
print('Crude (Normal) Confidence interval for gamma0 from Bootstrap results')
c(Bootstrap.L.gamma0 , Bootstrap.U.gamma0 )
# Crude (Normal) Confidence interval for gamma1 from Bootstrap results
print('Crude (Normal) Confidence interval for gamma1 from Bootstrap results')
c(Bootstrap.L.gamma1 , Bootstrap.U.gamma1 )
print('These are our original estimates. Clearly, they fall within the CIs above')
enzymeBoot$t0
# Gamma0
print('For Gamma0')
boot.ci(enzymeBoot, conf = .95, index = 1, type = 'basic')
# Gamma1
print('For Gamma1')
```

```
boot.ci(enzymeBoot, conf = .95, index = 2, type = 'basic')
############ QUESTION 5 begins here ###################
set.seed(32)
library(boot)
enzyme = read.csv('HW1_data.csv', sep=',') # Read in data
enzyme = enzyme[1:2] # Remove empty columns
# First we define the function, which produces the statistics we are interested in
# In this case estimates for the gamma parameters
enzymeFit<-function(Z, i, gammaInitial, x_pred) { # Note the additional argument x_pred
   Zboot<-Z[i,]
   x<-Zboot[[2]];y<-Zboot[[1]]
   model <- function(gamma){
      yhat = gamma[1]*x / (gamma[2]+x)
      sum((y-yhat)^2) # Return SSE
      }
   out<-nlm(model,p=gammaInitial)
   gamma <-out$estimate #parameter estimates
   # This line has been added to modify the enzymeFit function and therefore the contents of the Bootst
   y_pred <- gamma[1]*x_pred / (gamma[2]+x_pred)
}
enzymeBoot = boot(enzyme, enzymeFit, R = 20000, gammaInitial = gammaStart, x_pred = 27)
enzymeBoot
# The SE for the prediction is
Bootstrap.SE.y_pred = sd(enzymeBoot$t)
Bootstrap.SE.y_pred
# Calculating the CI for the predictable part i.e. g(x, gamma)
boot.ci(enzymeBoot, conf = .95, type = c("norm","basic"))
# This is a plot for our predictions i.e. for y_hat
plot(enzymeBoot)
y_pred.SE = sqrt( var(enzymeBoot$t) + MSE ) # The standard error for a future observation
g.hat = enzymeBoot$t0 # This is our MLE estimate for the parameters
# (It's derived from the bootstrap object, but it is the same as the numerical derivation of the MLEs)
c(g.hat - qnorm(.975)*y_pred.SE, g.hat + qnorm(.975)*y_pred.SE)
############ QUESTION 6 begins here ###################
enzyme = read.csv('HW1_data.csv', sep=',') # Read in data
enzyme = enzyme[1:2] # Remove empty columns
enzyme.sqrt.fit = lm(y~sqrt(x), data = enzyme)
summary(enzyme.sqrt.fit)
y = enzyme[1]
x = enzyme[2]
n = nrow(y)

AIC.sqrt = -2*as.numeric(logLik(enzyme.sqrt.fit))/n + 2*2/n
AIC.sqrt # This is AIC for the newly proposed model with sqrt(x)
AIC.model = -2*logLik(out2)/n +2*2/n # Using the output from the nls(), because it works with logLik
AIC.model
############ QUESTION 7 begins here ###################
# Note, our seed has been set to 32 somewhere above.
# Function for creating random index shuffling
CVInd <- function(n,K) {  #n is sample size; K is number of parts; returns K-length list of indices for
   m<-floor(n/K)  #approximate size of each part
   r<-n-m*K
   I<-sample(n,n)  #random reordering of the indices
```

```
   Ind<-list()  #will be list of indices for all K parts
   length(Ind)<-K
   for (k in 1:K) {
      if (k <= r) kpart <- ((m+1)*(k-1)+1):((m+1)*k)
         else kpart<-((m+1)*r+m*(k-r-1)+1):((m+1)*r+m*(k-r))
      Ind[[k]] <- I[kpart]  #indices for kth part of data
   }
   Ind
}
names(enzyme.sqrt.fit$coefficients)<-NULL
betaStart = enzyme.sqrt.fit$coefficients # Getting the parameter estimates from lm() call as first gues
betaStart
Nrep<-20 #number of replicates of CV
K<-n  #K-fold CV on each replicate
n.models = 2 #number of different models to fit and compare
FitFun1 <- function(x1,p) p[1]*x1/(p[2]+x1)
FitFun2 <- function(x1,p) p[1]+p[2]*sqrt(x1)
n=nrow(enzyme)
y<-enzyme$y
yhat=matrix(0,n,n.models)
MSE<-matrix(0,Nrep,n.models)
for (j in 1:Nrep) {
  Ind<-CVInd(n,K)
  yhat=matrix(0,n,n.models)
  for (k in 1:K) {
     out<-nls(y~FitFun1(x,p),data=enzyme[-Ind[[k]],],start=list(p=gammaStart))
     yhat[Ind[[k]],1]<-as.numeric(predict(out,enzyme[Ind[[k]],]))
     out<-nls(y~FitFun2(x,p),data=enzyme[-Ind[[k]],],start=list(p=betaStart))
     yhat[Ind[[k]],2]<-as.numeric(predict(out,enzyme[Ind[[k]],]))
  } #end of k loop
  MSE[j,]=apply(yhat,2,function(x) sum((y-x)^2))/n
} #end of j loop
print('MSE from all 20 CV iterations')
MSE
print('Avg. MSE from all 20 CV iterations')
MSEAve<- apply(MSE,2,mean); MSEAve #averaged mean square CV error
print('MSE standard deviations')
MSEsd <- apply(MSE,2,sd); MSEsd   #SD of mean square CV error
print('R^2 for the two models')
r2<-1-MSEAve/var(y); r2  #CV r^2
########### QUESTION 8 begins here ###################
non.lin.residuals = y - yhat[,1]
lin.residuals = y - yhat[,2]
print('Non-linear Residuals')
non.lin.residuals
print('Linear Residuals')
lin.residuals
print(' This is x')
x = x[[1]]
x
par(mfrow=c(1,2))
plot(x, non.lin.residuals,xlab = 'x', ylab = 'Residuals', main = 'Residuals for Non-Linear Model')
plot(x, lin.residuals,xlab = 'x', ylab = 'Residuals', main = 'Residuals for Linear Model')
```