hw01

Samuel Swain

2023-01-20

Problem 1

$$Joint\ PDF \Rightarrow\ f(y;\mu,\sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}\sigma^n} e^{\frac{-1}{2\sigma^2}\Sigma_{i=1}^n (y_i-\mu)^2}$$

$$Likelihood\ function \Rightarrow\ ln(1) - (\frac{n}{2}ln(2\pi) + nln(\sigma)) - \frac{1}{2\sigma^2}\Sigma_{i=1}^n (y_i-\mu)^2$$

$$\frac{\partial(f(y;\mu,\sigma))}{\partial\sigma} = \frac{-n}{\sigma} + \frac{1}{\sigma^3}\Sigma_{i=1}^n (y_i-\mu)^2 \Rightarrow\ Set = 0$$

$$\Rightarrow\ \frac{\Sigma_{i=1}^n (y_i-\mu)^2}{\sigma^3} = \frac{n}{\sigma} \Rightarrow from\ class:\ \mu = \hat{y} \Rightarrow \sigma = \sqrt{\frac{\Sigma_{i=1}^n (y_i-\bar{y})^2}{n}}$$

Problem 2

2(a)

```
## Linear Model
##  ---
##   Function:
##    - y = 0.03 + 0.45 * x
##   Coefs:
##    - gamma_0: 29.62201
##    - gamma_1: 13.44881
```

2(b)

```
## NLM:
##  - Estimates: 28.13688 12.57428
```

```
## NLS:
##  - Estimates: 28.13705 12.57445
```

Problem 3

3(a)

```
## NLM:
##  ---
##   Fisher Info Matrix:
##   15.37455 -13.73842
##   -13.73842 13.92197
##   Standard Errors:
##    - gamma_0: 0.7418084
##    - gamma_1: 0.7795476
```

3(b)

```
## NLS:
##  ---
##   Covariance Matrix:
##   0.5299535 0.5202828
##   0.5202828 0.5822505
##   Standard Errors:
##    - gamma_0: 0.727979
##    - gamma_1: 0.7630534
```

The results agree with part a as they are very close.

3(c)

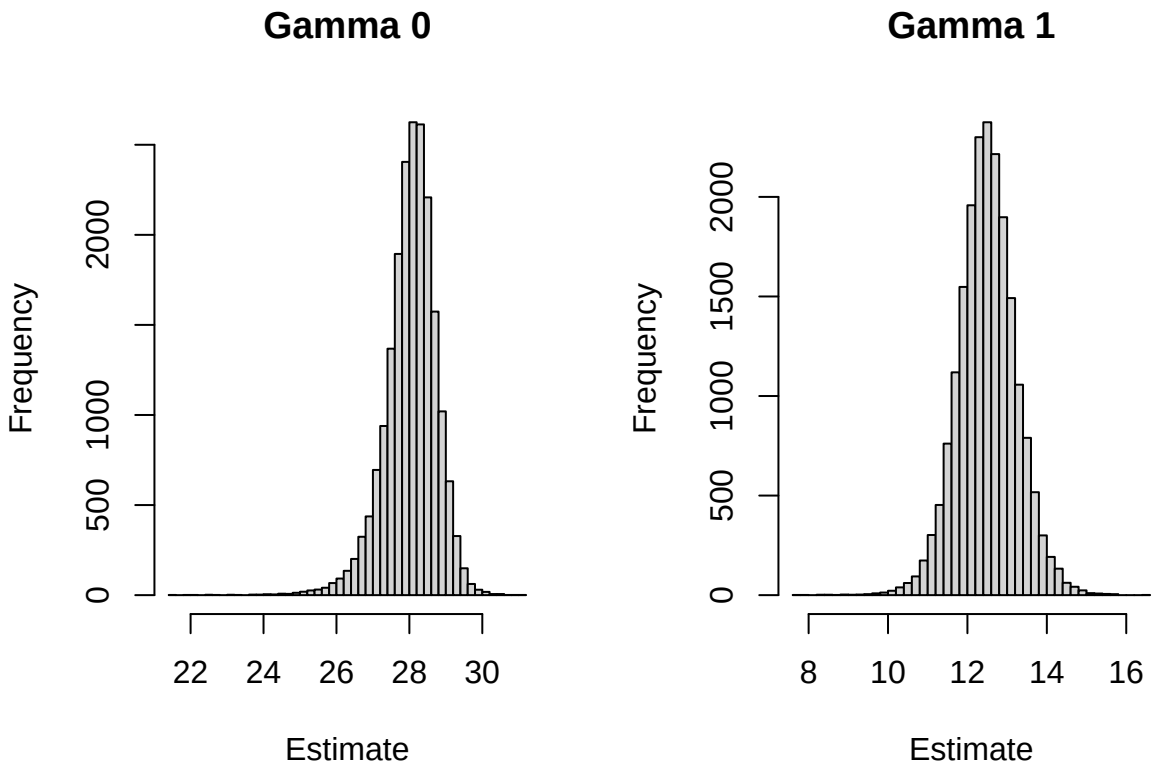
```
## NLM:
##  - gamma_0 CI: 26.68294 29.59083
##  - gamma_1 CI: 11.04637 14.10219
```

```
## NLS:
##  - gamma_0 CI: 26.71024 29.56386
##  - gamma_1 CI: 11.0789 14.07001
```

The two confidence intervals are almost the same.

Problem 4

4(a)




```
## Bootstrapped Standard Errors:
##  - gamma_0: 0.7103467
##  - gamma_1: 0.7359715
```

4(b)

```
## Crude 95% CI's:
##  - gamma_0: 26.64415 29.42871
##  - gamma_1: 11.05968 13.94469
```

4(c)

```
## Two-sided 95% CIs:
##  - gamma_0: 27.02675 29.82896
##  - gamma_1: 11.16271 14.09513
```

4(d)

They do agree. The histograms in part a suggest the gamma distributions are normally distributed. Because of this the "crude" CI calculated in part b will be roughly correct. This is why it is close to the CI calculated

using the boot.ci function in part c.

Question 5

```
## Future Response:
##  - L: 18.13702
##  - U: 20.34532

## Predictable CI:
##  - L: 18.90606
##  - U: 19.7199
```

I think the prediction interval better represents an interval that would contain future events. The prediction interval contains the confidence interval. Because of this it will always be larger to account for the extra noise. Because the interval is larger, it will better represent the interval we can be 95% sure will contain the prediction.

Question 6

```
## AIC:
##  - NLS Model: 1.739982
##  - Square Root Model: 2.947259
```

AIC suggests the old NSL model is better.

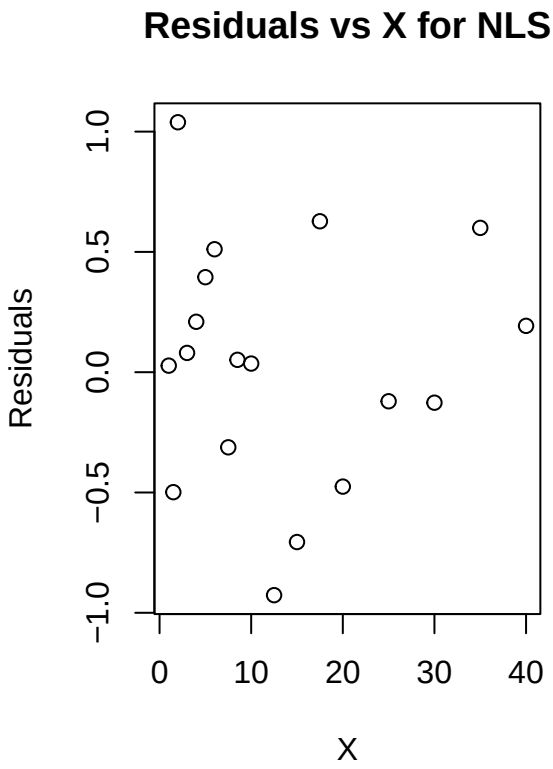
Question 7

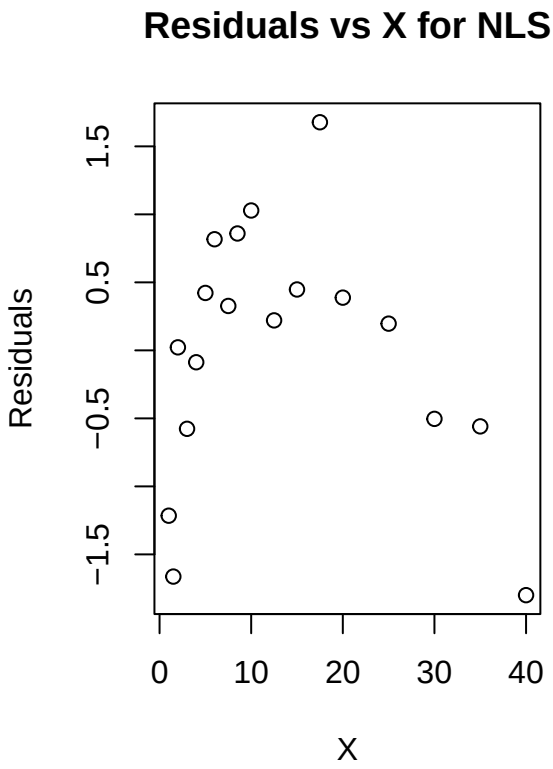
```
## NLS Model
##  - Average MSE: 0.3299942
##  - Standard Deviation of MSE: 0.05794925

## Square Root Model
##  - Average MSE: 1.19611
##  - Standard Deviation of MSE: 0.3717434
```

Cross validation suggests the NLS model performs better than the square root model.

Question 8





The residual plots for the NLS model seems more appropriate than the square root model. This is because the residuals for the NLS model show no pattern whereas the square root residual plot looks like a quadratic function. This agrees with out conclusions from questions 6 and 7 above.

Appendix

Commented out code is the output used to attain the results above. Commended to shorted final submission pdf.

Problem 2

```
knitr::opts_chunk$set(eval = FALSE) # Code in appendix doesn't need to run. Remove this if testing

library(readxl)
df_2 = read_excel("HW1_data.xls")
```

2(a) Code:

```
knitr::opts_chunk$set(eval = FALSE) # Code in appendix doesn't need to run. Remove this if testing
```

```r
fit_2a <- lm(1/y ~ 1/x, data = df_2)

gamma_0 = 1/fit_2a$coef[1]
gamma_1 = fit_2a$coef[2]/fit_2a$coef[1]

# cat("Linear Model", "\n", "---", "\n",
#     " Function:", "\n",
#     "  - y =", round(fit_2a$coef[1], 2), "+", round(fit_2a$coef[2], 2), "* x", "\n",
#     " Coefs:", "\n",
#     "  - gamma_0:", gamma_0, "\n",
#     "  - gamma_1:", gamma_1, "\n")
```

2(b) Code:

```r
knitr::opts_chunk$set(eval = FALSE) # Code in appendix doesn't need to run. Remove this if testing

x_1_2b <- df_2$x
y_2b <- df_2$y
p_est <- c(gamma_0, gamma_1)

# NLM
fn_nlm_2b <- function(p) {
  y_hat <- (p[1] * x_1_2b) / (p[2] + x_1_2b); return(sum((y_2b-y_hat)^2))
}
nlm_2b <- nlm(fn_nlm_2b,
              p=p_est,
              hessian=TRUE)
# cat("NLM:", "\n", "- Estimates:", nlm_2b$estimate, "\n")

# NLS
fn_nls_2b <- function(x_1, p) {
  (p[1] * x_1) / (p[2] + x_1)
}
nls_2b <- nls(y_2b ~ fn_nls_2b(x_1_2b, p),
              start=list(p=p_est),
              trace=FALSE)

nls_results <- nls_2b$m$getPars()

# cat("NLS:", "\n",
#     "- Estimates:", nls_results)
```

Problem 3

3(a) Code:

```r
knitr::opts_chunk$set(eval = FALSE) # Code in appendix doesn't need to run. Remove this if testing

MSE <- nlm_2b$minimum/(length(y_2b) - length(nlm_2b$estimate))
nlm_info_matrix <- nlm_2b$hessian/(2*MSE)
cov_theta_3a <- solve(nlm_info_matrix)
```

```
SE_3a <- sqrt(diag(cov_theta_3a))

# cat("NLM:", "\n", "---", "\n",
#     " Fisher Info Matrix:", "\n", "",
#     nlm_info_matrix[1:2], "\n", "",
#     nlm_info_matrix[3:4], "\n",
#     " Standard Errors:", "\n", "",
#     " - gamma_0:", SE_3a[1], "\n", "",
#     " - gamma_1:", SE_3a[2])
```

3(b) Code:

```
knitr::opts_chunk$set(eval = FALSE) # Code in appendix doesn't need to run. Remove this if testing

nls_cov_matrix <- vcov(nls_2b)
SE_3b <- sqrt(diag(nls_cov_matrix))

# cat("NLS:", "\n",  "---", "\n",
#     " Covariance Matrix:", "\n", "",
#     nls_cov_matrix[1:2], "\n", "",
#     nls_cov_matrix[3:4], "\n",
#     " Standard Errors:", "\n", "",
#     " - gamma_0:", SE_3b[1], "\n", "",
#     " - gamma_1:", SE_3b[2])
```

3(c) Code:

```
knitr::opts_chunk$set(eval = FALSE) # Code in appendix doesn't need to run. Remove this if testing

# NLM CI

# assign the first parameter estimate of 'nlm_2b' to 'nlm_estimate_1'
nlm_estimate_1 <- nlm_2b$estimate[1]

# assign the second parameter estimate of 'nlm_2b' to 'nlm_estimate_2'
nlm_estimate_2 <- nlm_2b$estimate[2]

# calculate lower and upper bounds of CI for the first parameter of 'nlm_2b'
CI_nlm_gamma_0 <- c(nlm_estimate_1 - 1.96 * SE_3a[1], nlm_estimate_1 + 1.96 * SE_3a[1])

# calculate lower and upper bounds of CI for the second parameter of 'nlm_2b'
CI_nlm_gamma_1 <- c(nlm_estimate_2 - 1.96 * SE_3a[2], nlm_estimate_2 + 1.96 * SE_3a[2])

# NLS CI

# calculate the CIs for all parameters of 'nls_2b' using 'confint.default' function
CIs_nls_3c <- confint.default(nls_2b)

# assign the first parameter estimate of 'nls_2b' to 'nls_estimate_1'
nls_estimate_1 <- CIs_nls_3c[c(1, 3)]
```

```r
# assign the second parameter estimate of 'nls_2b' to 'nls_estimate_2'
nls_estimate_2 <- CIs_nls_3c[c(2, 4)]

# calculate lower and upper bounds of CI for the first parameter of 'nls_2b'
CI_nls_gamma_0 <- c(nls_estimate_1[1], nls_estimate_1[2])

# calculate lower and upper bounds of CI for the second parameter of 'nls_2b'
CI_nls_gamma_1 <- c(nls_estimate_2[1], nls_estimate_2[2])

# output the calculated CIs for 'nlm_2b' to the console
# cat("NLM:", "\n",
#     "- gamma_0 CI:", CI_nlm_gamma_0, "\n",
#     "- gamma_1 CI:", CI_nlm_gamma_1, "\n")

# output the calculated CIs for 'nls_2b' to the console
# cat("NLS:", "\n",
#     "- gamma_0 CI:", CI_nls_gamma_0, "\n",
#     "- gamma_1 CI:", CI_nls_gamma_1, "\n")
```

Problem 4

4(a) Code:

```r
knitr::opts_chunk$set(eval = FALSE) # Code in appendix doesn't need to run. Remove this if testing

library(boot)

# Define a function that takes in a data set, a set of indices for re sampling, and an initial estimate
nlm_boot <- function(Z, i, theta0) {
  # Create a new data set by sub-setting the original using the set of indices
  Zboot <- Z[i,]
  # Assign variables x1 and y to the second and first columns of the re sampled data set, respectively
  x1 <- Zboot[[2]]; y <- Zboot[[1]]
  # Define a function that calculates the sum of squared errors between the predicted and actual values
  fn <- function(p) {y_hat <- (p[1] * x1) / (p[2] + x1); sum((y-y_hat)^2)}
  # Use the 'nlm' function to find the parameter estimates that minimize the sum of squared errors
  out <- nlm(fn, p=theta0)
  # Assign the parameter estimates to the variable 'theta'
  theta <- out$estimate
  # Return the parameter estimates
  return(theta)
}

# Use the 'boot' function to re sample the data set 'df_2' and apply the 'nlm_boot' function to each re
nlm_boot_4a <- boot(df_2, nlm_boot, 20000, theta0=p_est)
# Calculate the covariance matrix of the re sampled parameter estimates
nlm_cov_4a <- cov(nlm_boot_4a$t)
# Calculate the standard errors of the re sampled parameter estimates
SE_4a <- sqrt(diag(nlm_cov_4a))

# Set up a layout with two plots side-by-side
# par(mfrow = c(1, 2))
```

```
# Create a histogram of the re sampled estimates for the first parameter
# hist(nlm_boot_4a$t[, 1], breaks = 50, main = "Gamma 0", xlab = "Estimate")
# Create a histogram of the re sampled estimates for the second parameter
# hist(nlm_boot_4a$t[, 2], breaks = 50, main = "Gamma 1", xlab = "Estimate")

# Output the standard errors
# cat("Bootstrapped Standard Errors:", "\n",
#     "- gamma_0:", SE_4a[1], "\n",
#     "- gamma_1:", SE_4a[2])
```

4(b) Code:

```
knitr::opts_chunk$set(eval = FALSE) # Code in appendix doesn't need to run. Remove this if testing

# calculates lower and upper bounds of CI for first param in nlm_boot_4a
gamma_0_CI_4b <- c(mean(nlm_boot_4a$t[, 1]) - 1.96 * SE_4a[1], mean(nlm_boot_4a$t[, 1]) + 1.96 * SE_4a[

# calculates lower and upper bounds of CI for second param in nlm_boot_4a
gamma_1_CI_4b <- c(mean(nlm_boot_4a$t[, 2]) - 1.96 * SE_4a[2], mean(nlm_boot_4a$t[, 2]) + 1.96 * SE_4a[

# outputs calculated CIs with label indicating which parameter they correspond to
# cat("Crude 95% CI's:", "\n",
#     "- gamma_0:", gamma_0_CI_4b, "\n",
#     "- gamma_1:", gamma_1_CI_4b, "\n")
```

4(c) Code:

```
knitr::opts_chunk$set(eval = FALSE) # Code in appendix doesn't need to run. Remove this if testing

ci_4c_gamma_0 <- boot.ci(nlm_boot_4a, conf=c(.95), index = 1, type=c("basic"))
ci_4c_gamma_1 <- boot.ci(nlm_boot_4a, conf=c(.95), index = 2, type=c("basic"))

# cat("Two-sided 95% CIs:", "\n",
#     "- gamma_0:", ci_4c_gamma_0$basic[4:5], "\n",
#     "- gamma_1:", ci_4c_gamma_1$basic[4:5], "\n")
```

Question 5

Code:

```
knitr::opts_chunk$set(eval = FALSE) # Code in appendix doesn't need to run. Remove this if testing

# Define a function to perform a single iteration of the bootstrap method
boot_5a <- function(Z, i, theta0, x_pred) {
  # Select a subset of the data using the given indices
  Zboot <- Z[i,]
  # Assign the x and y values of the selected subset to x1 and y
  x1 <- Zboot[[2]]; y <- Zboot[[1]]
  # Define a function to calculate the sum of squared residuals
  fn <- function(p) {
```

```
    y_hat <- (p[1] * x1) / (p[2] + x1)
    sum((y-y_hat)^2)
  }
  # Use nonlinear least squares to estimate the parameters of the model
  out <- nlm(fn, p=theta0)
  theta <- out$estimate
  # Calculate the predicted value of y using the estimated parameters and the given value of x_pred
  y_pred <- (theta[1] * x_pred) / (theta[2] + x_pred)
}

# Perform bootstrap with 20000 re samples
nlm_boot_5 <- boot(df_2, boot_5a, R=20000, theta0=p_est, x_pred=c(27))

# Get y_hat initial and bootstrap values
y_hat_0_5 <- nlm_boot_5$t0
y_hat_boot <- nlm_boot_5$t

# Get the mean squared error
MSE_5 <- MSE

# Generate a normal distribution of errors
e_5 <- rnorm(nrow(y_hat_boot), mean = 0, sd = sqrt(MSE_5))

# subtract errors from predictions
y_boot <- y_hat_boot - e_5

# calculate lower and upper bounds of the 95% confidence interval
y_quant <- quantile(y_boot, prob = c(0.025, 0.975))
L_5 <- 2*y_hat_0_5 - y_quant[2]
U_5 <- 2*y_hat_0_5 - y_quant[1]
pi_5 <- c(L_5, U_5)
ci_5 <- boot.ci(nlm_boot_5, conf=c(.95), type=c("basic"))$basic[4:5]

# print out lower and upper bounds of the predictable and future response
# cat("Future Response:", "\n",
#     "- L:", pi_5[1],  "\n",
#     "- U:", pi_5[2], "\n")
# cat("Predictable CI:", "\n",
#     "- L:", ci_5[1],  "\n",
#     "- U:", ci_5[2])
```

Question 6

Code:

```
knitr::opts_chunk$set(eval = FALSE) # Code in appendix doesn't need to run. Remove this if testing

# assign number of rows in 'df_2' to 'n'
n <- nrow(df_2)

# calculate AIC for linear model 'fit_2a'
AIC_model_2a <- (-2*as.numeric(logLik(nls_2b))/n)+(2*3/n)
```

```r
# fit new model to data using sqrt of 'x' as predictor
fit_new <- lm(y ~ sqrt(x), data = df_2)
# calculate AIC for new model 'fit_new'
AIC_model_new <- (-2*as.numeric(logLik(fit_new))/n)+(2*3/n)

# output AIC values of both models
# cat("AIC:", "\n",
#     "- NLS Model:", AIC_model_2a, "\n",
#     "- Square Root Model:", AIC_model_new)
```

Question 7

Code:

```r
knitr::opts_chunk$set(eval = FALSE) # Code in appendix doesn't need to run. Remove this if testing

# K-fold indices
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
}

cv_folds_7 <- CVInd(length(x_1_2b), 3)

Nrep<-20 #number of replicates of CV
K<-3  #K-fold CV on each replicate
n.models = 2 #number of different models to fit and compare
n=length(x_1_2b)
y<-y_2b
yhat=matrix(0,n,n.models)
MSE<-matrix(0,Nrep,n.models)
for (j in 1:Nrep) {
  Ind<-CVInd(n,K)
  for (k in 1:K) {

     #the first model to compare
     out<-nls(y ~ fn_nls_2b(x, p),
              data = df_2[-Ind[[k]],],
              start=list(p=p_est))
     yhat[Ind[[k]],1]<-as.numeric(predict(out,df_2[Ind[[k]],]))

     #the second model to compare
     out<-lm(y ~ sqrt(x), data = df_2[-Ind[[k]],])
```

```
    yhat[Ind[[k]],2]<-as.numeric(predict(out,df_2[Ind[[k]],]))

  } #end of k loop
  MSE[j,]=apply(yhat,2,function(x) sum((y-x)^2))/n
} #end of j loop
# MSE
MSEAve <- apply(MSE,2,mean) # averaged mean square CV error
MSEsd <- apply(MSE,2,sd) # SD of mean square CV error
r2<-1-MSEAve/var(y) # CV r^2

# Print results nicely
# cat("NLS Model", "\n",
#     "- Average MSE:", MSEAve[1], "\n",
#     "- Standard Deviation of MSE:", MSEsd[1], "\n")
# cat("Square Root Model", "\n",
#     "- Average MSE:", MSEAve[2], "\n",
#     "- Standard Deviation of MSE:", MSEsd[2], "\n")
```

Question 8

Code:

```
knitr::opts_chunk$set(eval = FALSE) # Code in appendix doesn't need to run. Remove this if testing

# fit the first model
model1 <- nls_2b

# fit the second model
model2 <- fit_new

# create a data frame of the residuals
residuals1 <- data.frame(x = df_2$x, y = residuals(model1))
residuals2 <- data.frame(x = df_2$x, y = residuals(model2))

# Set up the layout for 2 plots in the same space
# par(mfrow = c(1, 2))

# create the first plot of residuals vs x for the first model
# plot(residuals1$x, residuals1$y, xlab = "X", ylab = "Residuals", main = "Residuals vs X for NLS")

# create the second plot of residuals vs x for the second model
# plot(residuals2$x, residuals2$y, xlab = "X", ylab = "Residuals", main = "Residuals vs X for NLS")
```