Homework 8 MATH531

Jared Andreatta

2025-03-13

Getting started

Loading the AudiA4 data and creating the X matrix for "broken" line regression.

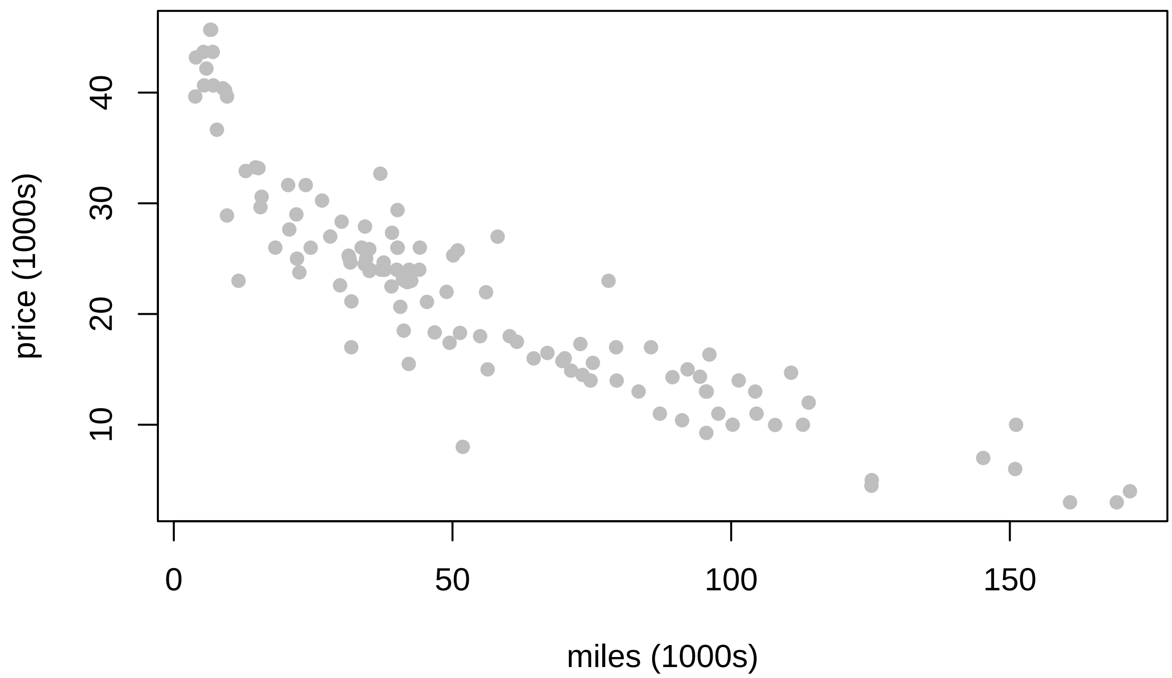
```
library( fields) # load fields package
```

```
## Warning: package 'fields' was built under R version 4.4.2

## Loading required package: spam

## Warning: package 'spam' was built under R version 4.4.2

## Spam version 2.11-1 (2025-01-20) is loaded.
## Type 'help( Spam)' or 'demo( spam)' for a short introduction
## and overview of this package.
## Help for individual functions is also obtained by adding the
## suffix '.spam' to the function name, e.g. 'help( chol.spam)'.

##
## Attaching package: 'spam'

## The following objects are masked from 'package:base':
##
##     backsolve, forwardsolve

## Loading required package: viridisLite

##
## Try help(fields) to get started.
```

```
setwd("C:\\Users\\Jared\\OneDrive\\Projects\\MinesProjects\\linear_model_theory_assignments")
load("AudiA4.rda" )
head( AudiA4)
```

```
##     year price mileage distance
## 58  2020 39649    3848       29
## 145 2020 43175    3962        7
## 10  2020 43675    5316        7
## 52  2020 40649    5417       29
## 143 2020 42175    5846        7
## 9   2020 45675    6539        7
```

```r
# change units so the numbers are simpler
price<- AudiA4$price/1000 # in thousands of dollars
mileage<- AudiA4$mileage/1000 # thousands of miles

# the data
plot(mileage, price, col="grey", pch=16,
  xlab= "miles (1000s)", ylab="price (1000s)")
```



Problem 1

Here is a standard OLS fit to these data. (You might want to review the handy "I" syntax in an lm formula.)

```r
OLSFit<- lm( price~ mileage + I(mileage^2))
summary( OLSFit)
```

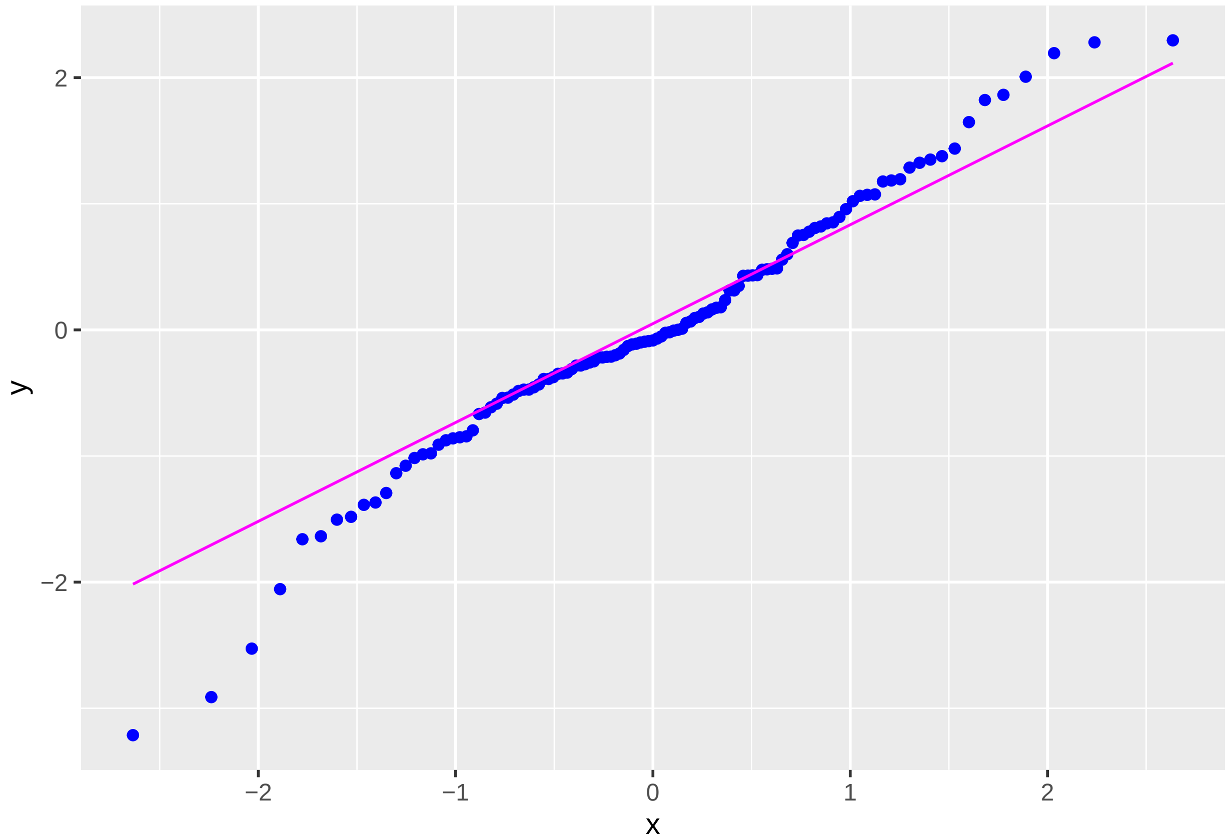
```
##
## Call:
## lm(formula = price ~ mileage + I(mileage^2))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -12.7513  -1.9017  -0.3323   2.2942   9.1094
##
```

```
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)  39.4106402  0.9287195  42.435  < 2e-16 ***
## mileage      -0.4350211  0.0299689 -14.516  < 2e-16 ***
## I(mileage^2)  0.0014482  0.0001927   7.514  1.3e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.968 on 116 degrees of freedom
## Multiple R-squared:  0.8422, Adjusted R-squared:  0.8395
## F-statistic: 309.6 on 2 and 116 DF,  p-value: < 2.2e-16
```

(a) Use a qqplot to assess if the standarized residuals follow a normal distribution, N(0,1) To do this from base R it is

```
n<- length(OLSFit$residuals)
p<- length(OLSFit$coefficients)
sigma2Hat<- sum(OLSFit$residuals^2)/ ( n-p)
theoretical<- qnorm(  ((1:n) -.5)/n )

plot( theoretical, sort(OLSFit$residuals/sqrt(sigma2Hat))
)
abline( 0,1, col="magenta")
```

```r
library(ggplot2)

resid <- OLSFit$residuals
n<- length(OLSFit$residuals)
p<- length(OLSFit$coefficients)
sigma2Hat<- sum(OLSFit$residuals^2)/ ( n-p)
theoretical<- qnorm(  ((1:n) -.5)/n )
std_resid <- resid / sqrt(sigma2Hat)

ggplot(data = data.frame(std_resid = std_resid), aes(sample = std_resid)) +
  stat_qq(color = "blue") +
  stat_qq_line(color = "magenta")
```

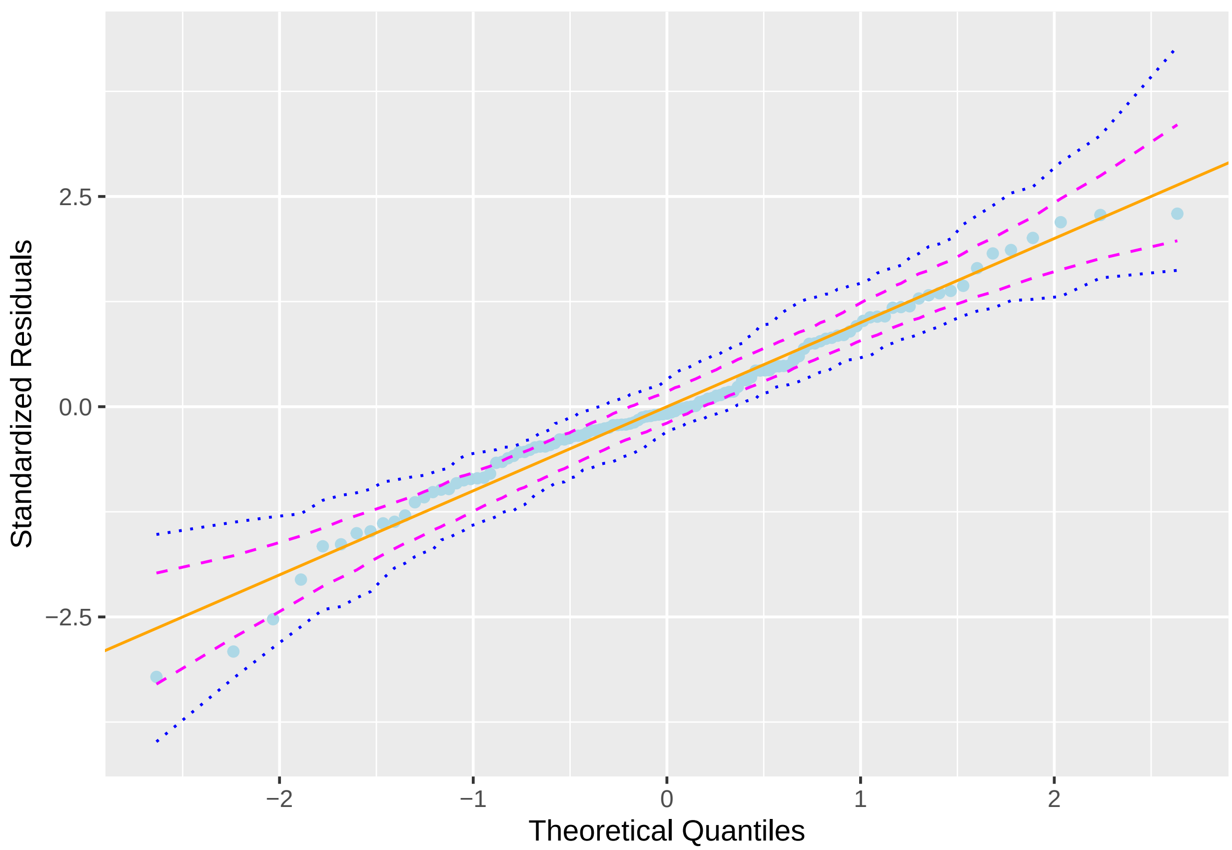


As part of your assessment add 90% bounds at each theoretical quantile for what you would expect if the data was drawn from iid N(0,1).

Hint: These 90% intervals will involve generating Monte Carlo samples and then finding 5% and 95% quantiles. Also generate simultanoeus intervals using the Bonferroni adjustment: **.05/n** and ** 1- .05/n **.

```
set.seed(42)

# Setup
m<-1000
n<-length(resid)

# Initializing empty matrix
simulated_quantiles <- matrix(NA, nrow = m, ncol = n)

# Monte Carlo samples
for (i in 1:m) {
  sample <- rnorm(n)
  simulated_quantiles[i, ] <- sort(sample)
}

# Obtaining 95% and 5% quantiles
lower <- apply(simulated_quantiles, 2, quantile, probs = 0.05)
upper <- apply(simulated_quantiles, 2, quantile, probs = 0.95)

# Bonferroni Adjustment
lower_bonf <- apply(simulated_quantiles, 2, quantile, probs = 0.05 / n)
```

```r
upper_bonf <- apply(simulated_quantiles, 2, quantile, probs = 1 - 0.05 / n)

# Initializing dataframe for plotting
df <- data.frame(
  theoretical = theoretical,
  std_resid = sort(std_resid),
  lower = lower,
  upper = upper,
  lower_bonf = lower_bonf,
  upper_bonf = upper_bonf
)

# Plotting
library(ggplot2)
ggplot(df, aes(x = theoretical, y = std_resid)) +
  geom_point(color = "lightblue") + # Residuals
  geom_abline(intercept = 0, slope = 1, color = "orange") + # Fitted Line
  geom_line(aes(y = lower), linetype = "dashed", color = "magenta") + # 5% bound
  geom_line(aes(y = upper), linetype = "dashed", color = "magenta") + # 95% bound
  geom_line(aes(y = lower_bonf), linetype = "dotted", color = "blue") + # Lower Bonferroni
  geom_line(aes(y = upper_bonf), linetype = "dotted", color = "blue") + # Upper Bonferroni
  labs(x = "Theoretical Quantiles",
       y = "Standardized Residuals")
```



Problem 2

Create a set of basis functions as bumps with the form:

$$H(d) = (1+d)e^(-d)$$

and the i^th basis function being

$$\phi_i(u) = H((u - v_i)/\alpha)$$

where $\{v_i\}$ are a grid of values and α a scaling parameter. I use $\alpha = 10$ below, don't change that.

Use the code below to create a matrix where all the basis functions are evaluated at all the mileages.

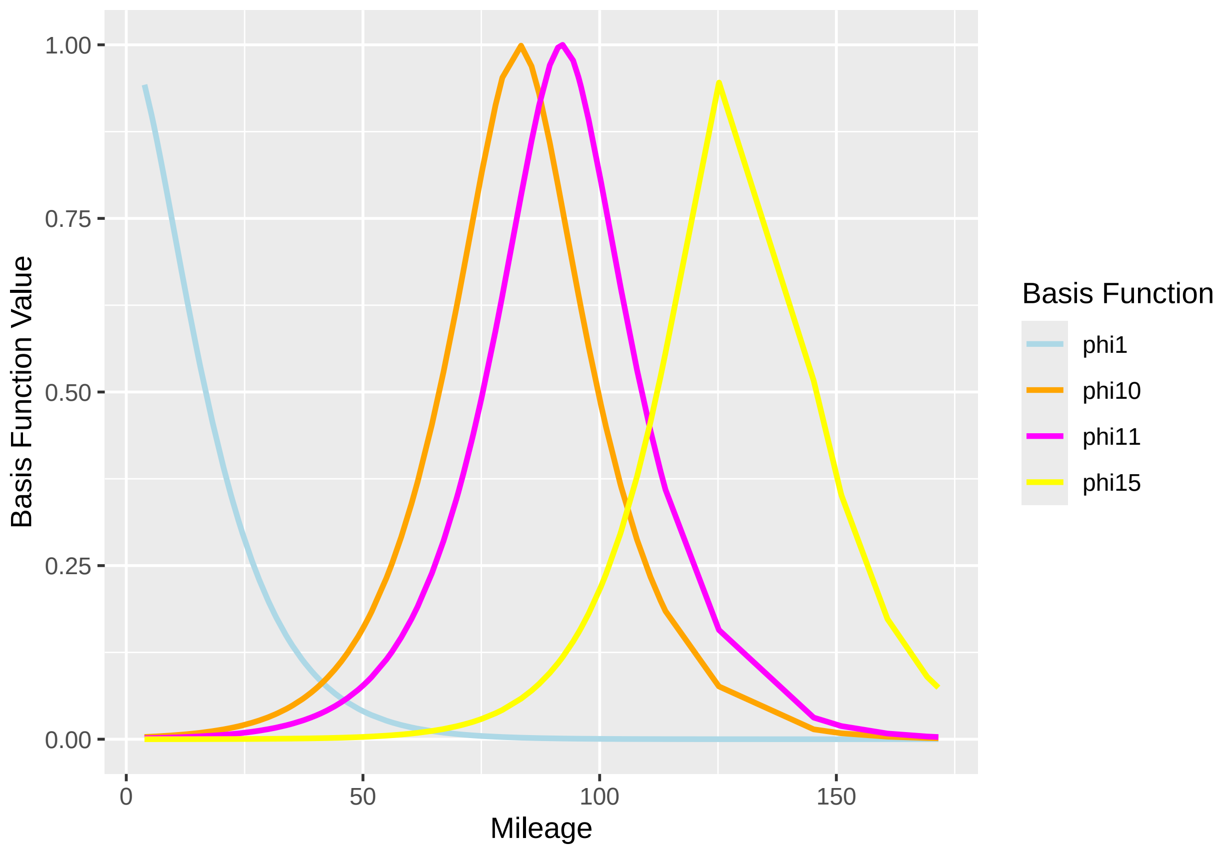
```
mGrid1<- seq( 0, 175, length.out=20)
bigD<- rdist( mileage, mGrid1)/10
# 10 is a scaling  for the width of the bumps
Phi<- (1+ bigD)* exp( -bigD)
```

2(a) There are 20 basis functions in this example. Plot basis functions, 1, 10,11, and 15 over the range of the mileage. Put these on the same figure for comparison.

Hint: These are the columns of Φ.

```
# Df of basis functionjs
df <- data.frame(
  mileage = mileage,
  phi1 = Phi[, 1],
  phi10 = Phi[, 10],
  phi11 = Phi[, 11],
  phi15 = Phi[, 15]
)

# Plotting basis functions
ggplot(df, aes(x = mileage)) +
  geom_line(aes(y = phi1, color = "phi1"), size = 1) +
  geom_line(aes(y = phi10, color = "phi10"), size = 1) +
  geom_line(aes(y = phi11, color = "phi11"), size = 1) +
  geom_line(aes(y = phi15, color = "phi15"), size = 1) +
  labs(x = "Mileage",
       y = "Basis Function Value",
       color = "Basis Function") +
  scale_color_manual(values = c("phi1" = "lightblue", "phi10" = "orange",
                                "phi11" = "magenta", "phi15" = "yellow"))
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

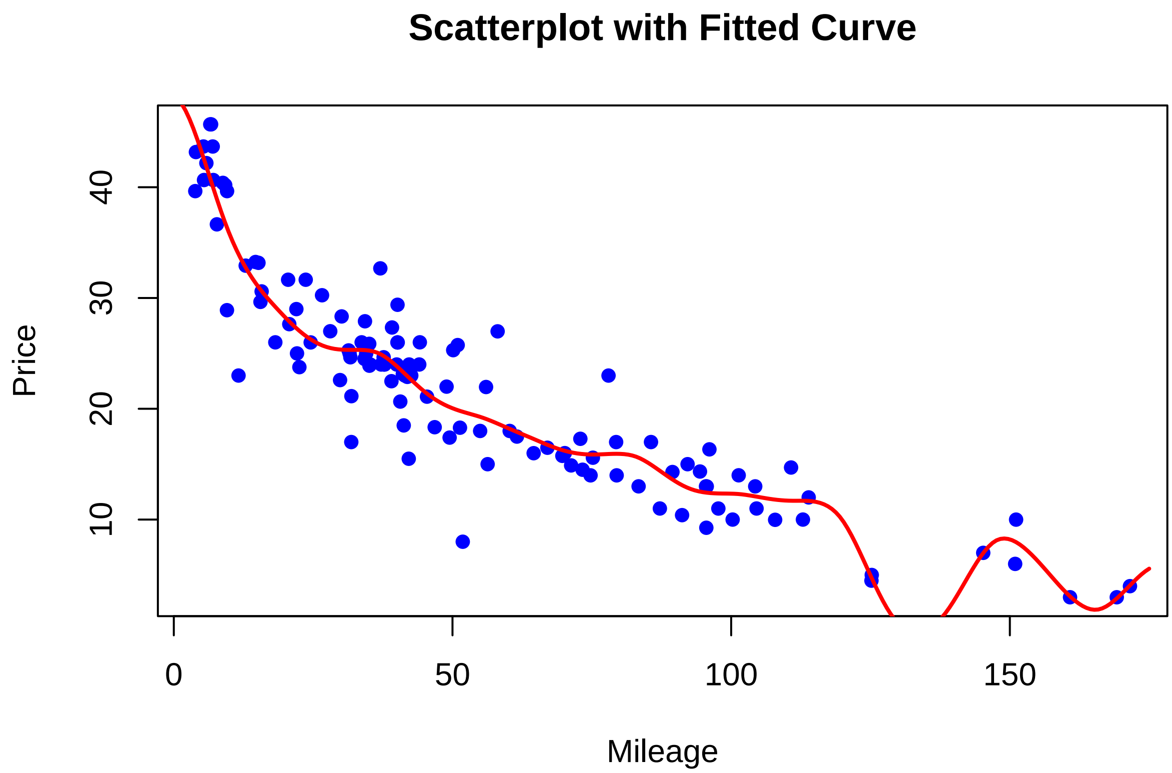


2(b) Fit an OLS model according to

```
FitBasis<- lm( price ~ Phi -1 )
```

Make a scatterplot of the data and add this fitted curve to it. Plot the estimated curve at the finer grid of points `mGrid2<- seq( 0, 175, length.out= 250)`

```
mGrid2 <- seq(0, 175, length.out = 250)

# New functions
bigD_new <- rdist(mGrid2, mGrid1) / 10
Phi_new <- (1 + bigD_new) * exp(-bigD_new)

# Price estimate with new basis functions
price_hat <- as.vector(Phi_new %*% coef(FitBasis))

# Plotting
plot(mileage, price, main = "Scatterplot with Fitted Curve",
     xlab = "Mileage", ylab = "Price", pch = 16, col = "blue")
lines(mGrid2, price_hat, col = "red", lwd = 2)
```



2(c) Now consider a ridge regression estimator:

$$\hat{\beta} = (\Phi^T \Phi + \alpha I)^{-1} \Phi^T y$$

where $\alpha \geq 0$ and y in this case is the price. Predicted values are

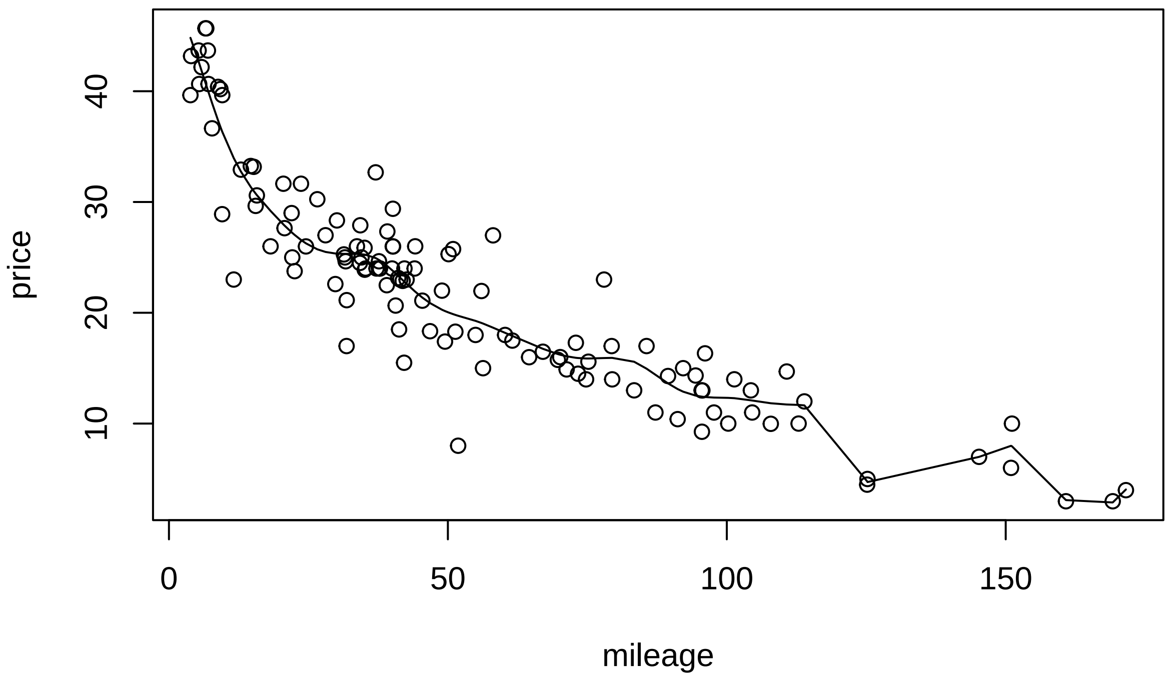
$$\hat{y} = \Phi \hat{\beta}$$

Below is a handy function to do this. Note that it is hardwired for this data set and basis.

```
mySmoother<-function(alpha){
smootherCoef<- solve(t(Phi)%*%Phi + alpha* diag(1,20))%*%t(Phi)%*%price
ridgeFit<-  Phi%*%smootherCoef
# note smoother Matrix is
# S<- Phi%*%solve(t(Phi)%*%Phi + alpha* diag(1,20))%*%t(Phi)
return( ridgeFit)
}
```

and as a code example

```
fitTest<- mySmoother(.0001)
plot( mileage,price)
lines( mileage, fitTest)
```



Vary `alpha` and choose a value that subjectively looks like a good fit to these data. Add this curve to your figure in 2(b). Also report the "effective number of parameters" in your choice as the trace of the matrix `smootherMatrix`. (You will have to adapt/hack the code for the `mySmoother` function to get this.)

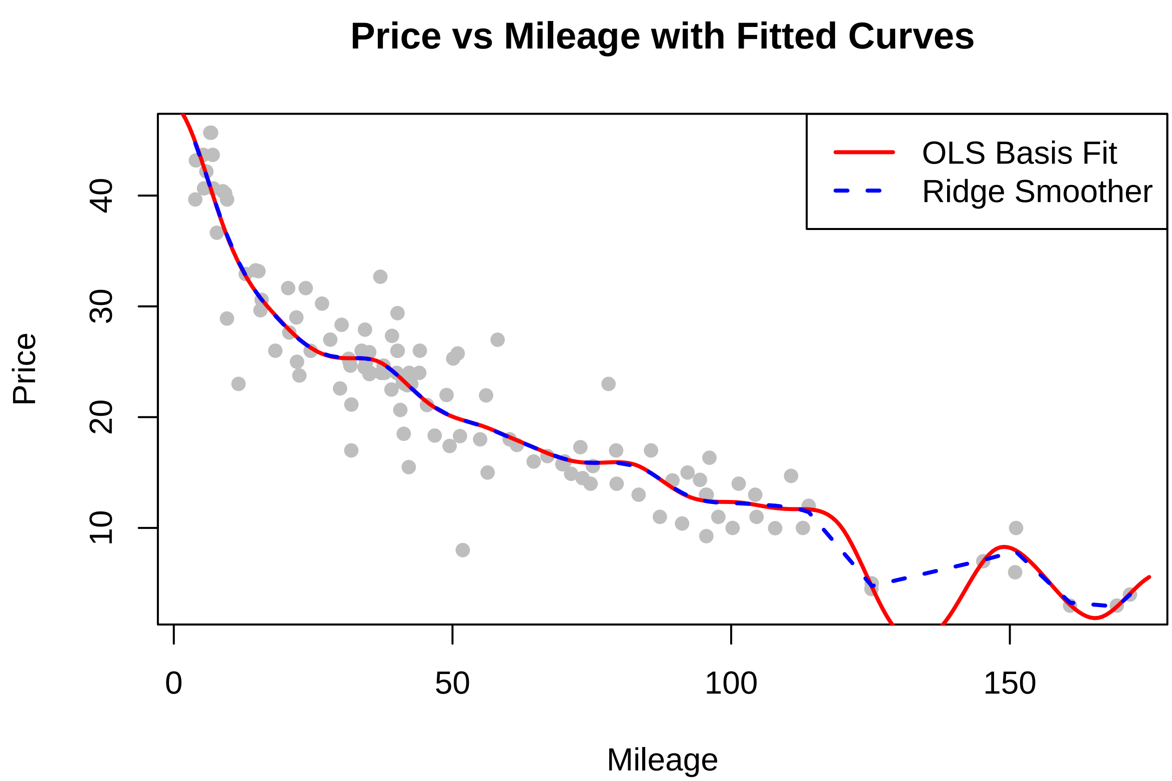
Hint: It is useful to vary α *equally spaced on a log scale* to get different amounts of smoothing. I used an alpha range of 1e-6 to 1e2.

```
# Modifying mySmoother function
mySmoother <- function(alpha) {
  smootherCoef <- solve(t(Phi) %*% Phi + alpha * diag(1, ncol(Phi))) %*% t(Phi) %*% price
  ridgeFit <- Phi %*% smootherCoef
  S <- Phi %*% solve(t(Phi) %*% Phi + alpha * diag(1, ncol(Phi))) %*% t(Phi)
  eff_params <- sum(diag(S))  # effective number of parameters
  return(list(fit = ridgeFit, eff_params = eff_params))
}

alpha <- .001
result <- mySmoother(alpha)
fitRidge <- result$fit
eff_params <- result$eff_params
cat("Effective number of parameters (trace(S)): ", round(eff_params, 2), "\n")
```

```
## Effective number of parameters (trace(S)):  18.35
```

```
# Scatterplot
plot(mileage, price, main="Price vs Mileage with Fitted Curves",
```

```r
     xlab="Mileage", ylab="Price", pch=16, col="grey")

# OLS Curve
lines(mGrid2, price_hat, col="red", lwd=2)

# Ridge Curve
ord <- order(mileage)
lines(mileage[ord], fitRidge[ord], col="blue", lwd=2, lty=2)

legend("topright", legend=c("OLS Basis Fit", "Ridge Smoother"),
       col=c("red", "blue"), lwd=2, lty=c(1,2))
```



EXTRA CREDIT: Explain how to modify this estimator to include a constant and linear function where as $\alpha \to \infty$ the ridge estimate is just the OLS estimate of a line. (This should help with getting a better fit at the ends.)

Problem 3

This problem compares the OLS fit quadratic function to the Bayesian version. To make the uncertainty of the parameters more comparable in size use the X matrix:

```r
X<- cbind( 1, mileage/10, (mileage^2)/1000 )
```

Because this is a linear transformation you should get the same predicted values and the inferences will be the same. Of course the coefficients are different.

```r
OLSFit2<- lm( price ~ X -1)
summary(OLSFit2)$coefficients
```

```
##     Estimate Std. Error    t value     Pr(>|t|)
## X1 39.410640  0.9287195  42.435462 1.704583e-72
## X2 -4.350211  0.2996893 -14.515738 7.575508e-28
## X3  1.448150  0.1927143   7.514493 1.298652e-11
```

```r
ols_coef <- coef(OLSFit2)
ols_se <- summary(OLSFit2)$coefficients[, "Std. Error"]
```

Below is an excerp from the wikipedia page on the Bayesian linear model that details the posterior distribution. For the priors applied to the quadratic regression model, set $\mu_0 = 0$ $\Lambda_0 = .01$ and for the Inverse gamma prior on σ^2 use $a_0 = 1/2$ and $b_0 = (1/2) * 20$. These will give a prior distribution around 20 with a large spread.

```r
# Set priors
mu0 <- rep(0, 3)
Lambda0 <- 0.01 * diag(3)
a0 <- 1/2
b0 <- 0.5 * 20
```

Now sample from the joint posterior 10000 times. That is for 10000 repeatitions first sample σ^2 from its posterior IG distribution (IG(a_n, b_n)) and then sample β from a multivariate normal conditional on the value sampled for σ^2 Find the mean and standard deviations for the three regression parameters from these 10000 samples and compare them to the OLS estimates and standard errors obtained from the OLS fit above.

```r
# Posterior calculations:
Vn <- solve(t(X) %*% X + Lambda0)
mun <- Vn %*% (t(X) %*% price + Lambda0 %*% mu0)
an <- a0 + n/2
bn <- b0 + 0.5 * ( t(price) %*% price - t(mun) %*% solve(Vn) %*% mun )

# Set up sampling
niter <- 10000
beta_samples <- matrix(NA, niter, 3)
sigma2_samples <- rep(NA, niter)

# For mvrnorm
library(MASS)
```

```
## Warning: package 'MASS' was built under R version 4.4.2
```

```r
for(i in 1:niter){
  # Sample sigma2 from IG(an, bn): 1/sigma2 ~ Gamma(an, rate=bn)
  sigma2_samples[i] <- 1 / rgamma(1, shape = an, rate = bn)
  # Sample beta from N(mun, sigma2 * Vn)
  beta_samples[i, ] <- mvrnorm(1, mun, sigma2_samples[i] * Vn)
}
```

```r
# Compute posterior means and sds
posterior_means <- colMeans(beta_samples)
posterior_sds <- apply(beta_samples, 2, sd)

# Prepare a comparison table:
comparison <- data.frame(
  Parameter = c("Intercept", "Mileage", "Mileage^2"),
  OLS_Estimate = round(ols_coef, 4),
  OLS_SE = round(ols_se, 4),
  Bayesian_Mean = round(posterior_means, 4),
  Bayesian_SD = round(posterior_sds, 4)
)

print(comparison)
```

```
##    Parameter OLS_Estimate OLS_SE Bayesian_Mean Bayesian_SD
## X1 Intercept      39.4106 0.9287       39.3837      0.9345
## X2   Mileage      -4.3502 0.2997       -4.3437      0.3027
## X3 Mileage^2       1.4482 0.1927        1.4450      0.1945
```