Statistical Methods - HW3

2023-02-02

Problem 1

(Is polynomial regression well-conditioned?) Consider the canonical design matrix for fitting a polynomial of degree p based on $x_1, ..., x_n$, meaning, $X = (x_i{}^j)$ for i = 1, . . . , n and j = 0, . . . , p. Suppose the x_i's are evenly distributed in (0, 1), for example, $x_i = i/(n + 1)$ for i = 1, . . . , n. For each $p \in 1, ..., 20$ and each $n \in 30, 50, 100, 200, 500, 1000$, compute the condition number of the design matrix. Produce a useful plot for visualizing the result of these computations. Offer brief comments.

Condition number of a given matrix is defined as a measure of its sensitivity to perturbations in the input data.

The condition number with a high value indicates that small changes in the input data can cause large changes in the output, making the solution to the linear regression problem less stable.

In polynomial regression, the condition number of the design matrix is defined as the ratio of the largest and smallest singular values of the matrix.

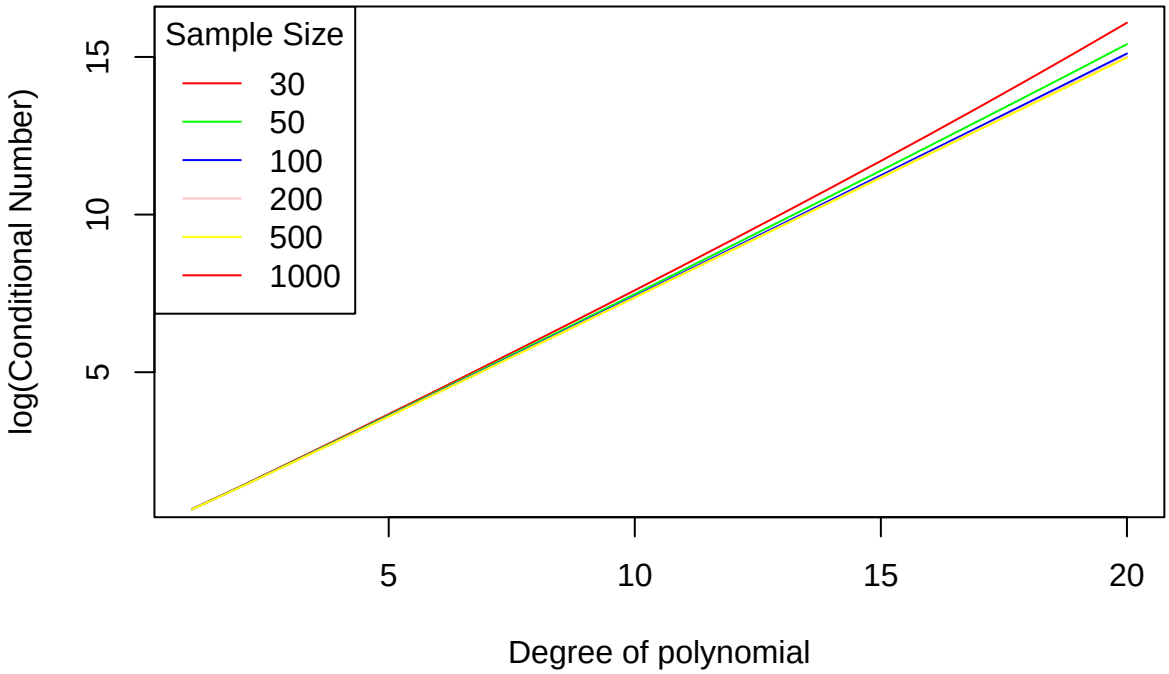
```
# Degrees of the polynomials
p_values = 20
deg = 1:p_values
# Sample sizes
n_values<-list(30,50,100,200,500,1000)
clr = c("red","green","blue","pink","yellow")
c = 1

plot(x=NA, y=NA, xlim=c(1,20), ylim=c(1,16),
     xlab="Degree of polynomial", ylab="log(Conditional Number)")

# Computing the condition number for each sample size and degree
for (n in n_values) {
  # List to contain the condition numbers
  e <- list()
  for(p in 1:p_values) {
    # Splitting into n evenly distributed points
    x <- seq(1/(n+1), n/(n+1), by=1/(n+1))
    # Fitting the x into a polynomial with degree p
    X = poly(x, degree=p, raw=TRUE)
    # Adding 1s to the polynomial
    X = cbind(rep(1, n), X)
    # Computing svd to find eigen values
    d = svd(X)$d
    # Appending condition number to the list
    e <- append(e, max(d)/min(d))
```

```
  }
  e <- as.numeric(unlist(e))
  # Plot
  lines(deg, log10(e), col=clr[c])
  c = c + 1
}

# Adding a legend to the plot
legend("topleft", title = "Sample Size", legend=n_values, col=clr, lty=1)
```



The above code computes the condition number of the design matrix for polynomial regression with varying degrees (p) and sample sizes (n), and produces a plot of the log10 of the condition numbers with respect to the degree of the polynomial. The x-axis represents the degree of the polynomial, and the y-axis represents the condition number.

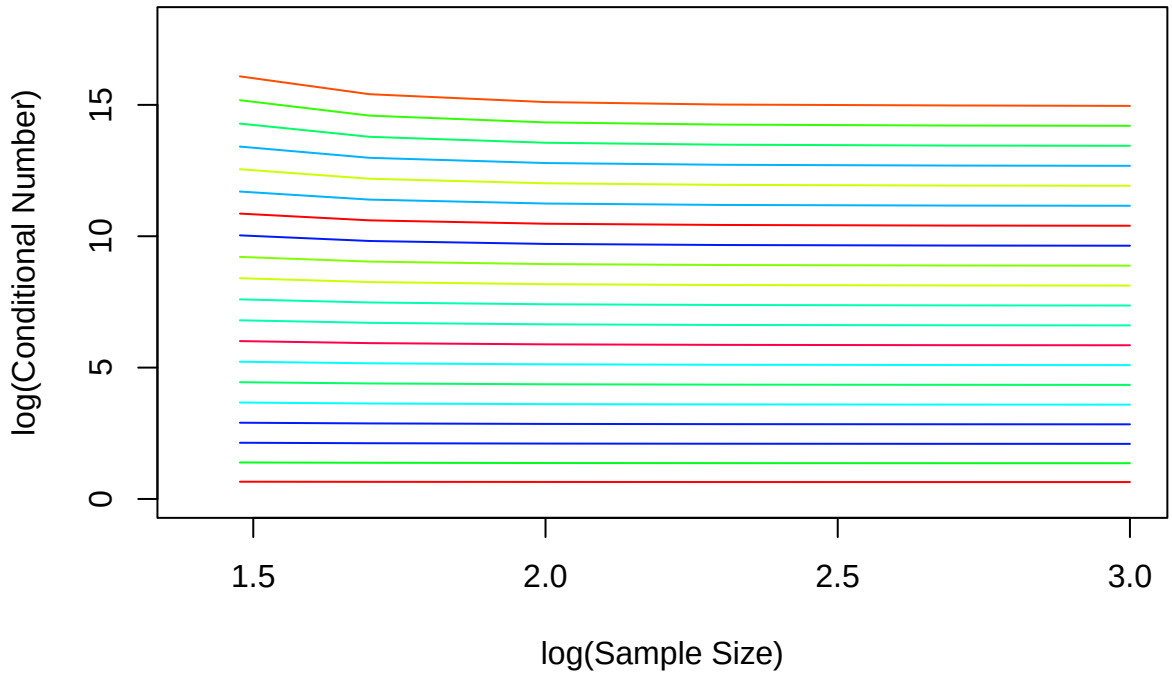
From this plot, we can see that as the degree of the polynomial increases, the condition number of the design matrix increases. As a result, even a small change in data makes the fit unreliable.

As the degree of the polynomial increases, the columns of the polynomial becomes more dependent and has eigen values closer to 0 and as a result the condition number increases.

```
# Degrees of the polynomials
p_values = 20
deg = 1:p_values
# Sample sizes
n_values<-list(30,50,100,200,500,1000)
df<-data.frame()
```

```
plot(x=NA, y=NA, xlim=c(1.4,3), ylim=c(0,18),
     xlab="log(Sample Size)", ylab="log(Conditional Number)")

# Computing the condition number for each degree and sample size
for (p in 1:p_values) {
  e <- list()
  for(n in n_values) {
    # Splitting into n evenly distributed points
    x <- seq(1/(n+1), n/(n+1), by=1/(n+1))
    # Fitting the x into a polynomial with degree p
    X = poly(x, degree=p, raw=TRUE)
    # Adding 1s to the polynomial
    X = cbind(rep(1, n), X)
    # Computing svd to find eigen values
    d = svd(X)$d
    # Appending condition number to the list
    e <- append(e, max(d)/min(d))
  }
  e <- as.numeric(unlist(e))
  nu <- as.numeric(unlist(n_values))
  # Plot
  lines(log10(nu), log10(e), col=sample(rainbow(p_values)))
  # Put values of e into the dataframe
  df = rbind(df, e)
}
```




```
df = df[-1,]
print(df)
```

```
##    X4.53568646889199 X4.47721843740548 X4.4339153494489 X4.41243517706191
## 2       2.444352e+01      2.380050e+01     2.333823e+01      2.311326e+01
## 3       1.379508e+02      1.322247e+02     1.282591e+02      1.263752e+02
```

```
## 4       7.965895e+02      7.495574e+02     7.183918e+02      7.040057e+02
## 5       4.673628e+03      4.302247e+03     4.068139e+03      3.963578e+03
## 6       2.779070e+04      2.491996e+04     2.320705e+04      2.246947e+04
## 7       1.673614e+05      1.454355e+05     1.331074e+05      1.280053e+05
## 8       1.020895e+06      8.545258e+05     7.667576e+05      7.319387e+05
## 9       6.311780e+06      5.053131e+06     4.432898e+06      4.197576e+06
## 10      3.959027e+07      3.007029e+07     2.570980e+07      2.413083e+07
## 11      2.522447e+08      1.800896e+08     1.495429e+08      1.390067e+08
## 12      1.634859e+09      1.085664e+09     8.721836e+08      8.021799e+08
## 13      1.079654e+10      6.589861e+09     5.100050e+09      4.636542e+09
## 14      7.278685e+10      4.028804e+10     2.989761e+10      2.683722e+10
## 15      5.020082e+11      2.481784e+11     1.757037e+11      1.555434e+11
## 16      3.550661e+12      1.541083e+12     1.035157e+12      9.026000e+11
## 17      2.582544e+13      9.650962e+12     6.113975e+12      5.243547e+12
## 18      1.937971e+14      6.098709e+13     3.620239e+13      3.049088e+13
## 19      1.507725e+15      3.895918e+14     2.151082e+14      1.773077e+14
## 20      1.215983e+16      2.544406e+15     1.279214e+15      1.028800e+15
##    X4.39960097093609 X4.39533180782886
## 2       2.298020e+01      2.293617e+01
## 3       1.252752e+02      1.249134e+02
## 4       6.957352e+02      6.930369e+02
## 5       3.904540e+03      3.885454e+03
## 6       2.206136e+04      2.193080e+04
## 7       1.252445e+05      1.243715e+05
## 8       7.135453e+05      7.078036e+05
## 9       4.076408e+06      4.039105e+06
## 10      2.333944e+07      2.309939e+07
## 11      1.338720e+08      1.323389e+08
## 12      7.690418e+08      7.593091e+08
## 13      4.423586e+09      4.362110e+09
## 14      2.547349e+10      2.508679e+10
## 15      1.468359e+11      1.444120e+11
## 16      8.471478e+11      8.319982e+11
## 17      4.891387e+12      4.796833e+12
## 18      2.826460e+13      2.767133e+13
## 19      1.635493e+14      1.596120e+14
## 20      9.497844e+14      9.147359e+14
```

The above code computes the condition number of the design matrix for polynomial regression with varying degrees (p) and sample sizes (n), and produces a plot of the log10 of the condition numbers. The x-axis represents the log of sample sizes, and the y-axis represents the log of condition numbers.

From the plot and the values printed using the dataframe, we can see that as the sample size increases, the condition number decreases gradually and gets well conditioned. As a result, the fit becomes more reliable.

In conclusion, fitting a higher degree polynomial makes it ill-conditioned and increasing the sample size makes it well conditioned.

Problem 2 (Piecewise constant fit)

(a) Write a function piecewiseConstant(x, y, L, plot = TRUE) taking in a one dimensional predictor variable x with values in [0, 1] and a response y, and fits a piecewise constant model (by least squares) on 2^L intervals of equal length partitioning the unit interval (L is a nonnegative integer) in the form of a numerical vector of length 2^L, with the option of producing a scatterplot with the fit overlaid.

Assuming the values of x passed to the function are already normalised and in the range [0, 1]

```r
# piecewiseConstant function implementation
piecewiseConstant <- function(x, y, L, plot=TRUE) {
  # Splitting into 2^L intervals
  K = seq(0, 1, length.out = (2^L) + 1)
  # Initialising pts and val to 0
  pts = rep(0, 2^(L+1))
  val = rep(0, 2^(L+1))
  for (j in 1:2^L) {
    # Binary indicators for each interval
    I = (K[j] < x) & (x <= K[j+1])
    # Fit the piecewise constant model using least squares
    if(sum(I, na.rm=TRUE) != 0) {
      fit = lm(y[I] ~ 1)
    }
    # Assigning values to the fitted piecewise model
    pts[2*j-1] = K[j]
    pts[2*j] = K[j+1]
    val[2*j-1] = coef(fit)
    val[2*j] = coef(fit)
  }
  # Plotting the scatter plot with overlaid fit
  if(plot){
    plot(x, y)
    lines(pts, val, col="grey", lwd = 3)
  }
}
```

The above function takes in a one dimensional predictor variable x with values in [0, 1] and a response y, and fits a piecewise constant model (by least squares) on 2^L intervals of equal length partitioning the unit interval (L is a nonnegative integer) in the form of a numerical vector of length 2^L, with the option of producing a scatterplot with the fit overlaid.

(b) Apply your function to explaining City Mpg as a piecewise constant function of Horsepower in the 04cars dataset. Produce a single scatterplot, with lines corresponding to the fit with L = 2 (blue), L = 3 (green), and L = 4 (red). Add a legend, etc, so it looks 'nice'.

Made few changes to the function in Qn. 2(a) to satisfy the requirements of this question.

```
# Loading the 04cars data
load("04cars.rda")

# Creating a dataframe with required columns from the data
df = dat[,c(13,14)]
df = df[complete.cases(df),]
df = as.data.frame(df)
names(df) = c("hp", "citympg")
dat = df
attach(dat)

# piecewiseConstant function implementation for the given requirement
piecewiseConstant <- function(x, y, L, plot=TRUE) {
  # Splitting into 2^L intervals
  K = seq(0, 1, length.out = (2^L) + 1)
  # Initialising pts and val to 0
  pts = rep(0, 2^(L+1))
  val = rep(0, 2^(L+1))
  for (j in 1:2^L) {
    # Binary indicators for each interval
    I = (K[j] < x) & (x <= K[j+1])
    # Fit the piecewise constant model using least squares
    if(sum(I, na.rm=TRUE) != 0) {
      fit = lm(y[I] ~ 1)
    }
    # Assigning values to the fitted piecewise model
    pts[2*j-1] = K[j]
    pts[2*j] = K[j+1]
    val[2*j-1] = coef(fit)
    val[2*j] = coef(fit)
  }
  # Plot with overlaid fit
  if(plot){
    if(L == 2) {
      lines(pts, val, col="blue", lwd = 3)
    }
    else if(L == 3) {
      lines(pts, val, col="green", lwd = 3)
    }
    else if(L == 4) {
```

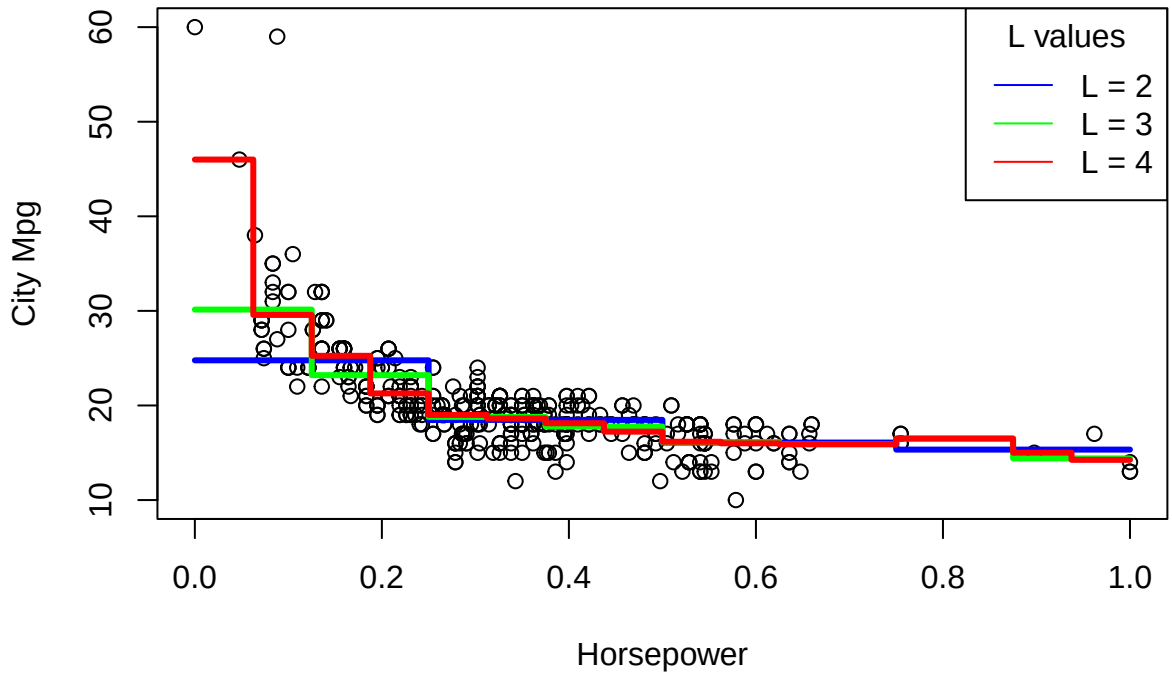
```
      lines(pts, val, col="red", lwd = 3)
    }
    else {
      lines(pts, val, col="grey", lwd = 3)
    }
  }
}

# Function to normalize data
Normalize_Data <- function(val) {
  return ((val - min(val)) / (max(val) - min(val)))
}

# Set the predictor variable x to horsepower
x <- Normalize_Data(dat$hp)
# Set the response variable y to city mpg
y <- dat$citympg

# Plot the scatter plot of the data
plot(x, y, xlab = "Horsepower", ylab = "City Mpg")

# Overlay the fits with L = 2, 3, and 4
for (i in 2:4) {
  piecewiseConstant(x, dat$citympg, i, TRUE)
}
# Adding a legend to the plot
legend("topright", title = "L values", legend=c("L = 2", "L = 3", "L = 4"),
       col=c("blue", "green", "red"), lty=1)
```



The above plot shows the piecewise plots corresponding to the fit with L = 2, L = 3 and L = 4.

As the value of L increases, the number of intervals increases and the size of the interval decreases.

Also, as the value of L increases, the data is better fit but it maybe prone to overfitting.

Contribution Statement:

1. Swetha Arunraj (PID: A59019948):

- Coded Problem 1
- Added comments to the code of Problem 1
- Added description/comments to Problem 1
- Contributed to embed the codes in R Markdown and create the final PDF

2. Harin Raja Radha Krishnan (PID: A59019874):

- Coded Problem 2
- Added comments to the code of Problem 2
- Added description/comments to Problem 2
- Contributed to embed the codes in R Markdown and create the final PDF