HW3

Q2

Initializing Environment

```r
# rm(list =ls())
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.6     v purrr   0.3.4
## v tibble  3.1.7     v dplyr   1.0.9
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(reshape2)
```

```
##
## Attaching package: 'reshape2'

## The following object is masked from 'package:tidyr':
##
##     smiths
```

Data Generating Functions

```r
#rm(list =ls())

lin <- function(x) {
  return(2*x)
}

sin_2pi <- function(x) {
  return(sin(2*pi*x))
}

sin_30 <- function(x) {
  return(sin(30*x))
}
```

Choosing bandwidth for LOESS to minimize MSE

```r
# Set seed for reproducibility
set.seed(42)

# Check the best bandwidth for LOESS with degree 0,1,2
```

```r
n = 1e4

# Use the most wiggly function to ensure most appropriate bandwidth
x <- sort(runif(n))
true_x <- sin_30(x)
eps <- rnorm(n)
y <- true_x+ eps
mse = c()
interval = seq(0.1,10,0.1)

# Degree 0
for (i in 1:100){
  model <- loess(y ~ x, span = (interval[i]) * n^(-1/3),method = c("loess"), degree =0)
  mse[i] = mean((predict(model, newdata = data.frame(x)) - true_x)^2)
}
# Here we select the bandwidth that minimizes the MSE: The one selected has coefficient 4
interval[which(mse == min(mse))]
```

```
## [1] 0.8
```

```r
# Degree 1
for (i in 1:100){
  model <- loess(y ~ x, span = (interval[i]) * n^(-1/3),method = c("loess"), degree =1)
  mse[i] = mean((predict(model, newdata = data.frame(x)) - true_x)^2)
}
# Here we select the bandwidth that minimizes the MSE: The one selected has coefficient 4
interval[which(mse == min(mse))]
```

```
## [1] 1
```

```r
# Degree 2
for (i in 1:100){
  model <- loess(y ~ x, span = (interval[i]) * n^(-1/5),method = c("loess"), degree =2)
  mse[i] = mean((predict(model, newdata = data.frame(x)) - true_x)^2)
}
# Here we select the bandwidth that minimizes the MSE: The one selected has coefficient 3
interval[which(mse == min(mse))]
```

```
## [1] 0.7
```

```r
#model <-loess(y ~ x, span = (interval[2]) * n^(-1/3),method = c("loess"))
#mse
```

Here I used the second data generating function to find the coefficient that minimizes MSE. The main part of the bandwidth is given by $h = n^{-1/(2k+1)}$. So for the degree 1 it is $n^{(-1/3)}$ and for degree 2 it is $n^{(-1/5)}$. After obtaining the coefficient I can use it in the following simulate function.

```r
simulate <- function(n, func) {
  #sampling x and epsilon
  #need to sort x because the kernel functions output values with ordered x!!
  x <- sort(runif(n))
  eps <- rnorm(n)

  #calculate y
  y <- func(x) + eps

  #fit linear and polynomial models
```

```
  y_1 <- lm(y~x)$fitted.values
  y_2 <- lm(y~poly(x,2))$fitted.values
  y_3 <- lm(y~poly(x,3))$fitted.values
  y_4 <- lm(y~poly(x,4))$fitted.values
  y_5 <- lm(y~poly(x,5))$fitted.values

  #fit LOESS models
  loess_0 <- loess(y ~ x, span =  0.8*n^(-1/3),method = c("loess"), degree = 0)
  loess0_pred <- predict(loess_0, newdata = data.frame(x))
  loess_1 <- loess(y ~ x, span =  n^(-1/3),method = c("loess"), degree = 1)
  loess1_pred <- predict(loess_1, newdata = data.frame(x))
  loess_2 <- loess(y ~ x, span =  0.7 * n^(-1/5),method = c("loess"), degree = 2)
  loess2_pred <- predict(loess_2, newdata = data.frame(x))

  #choice of kernel bandwidth
  h<- 0.5*n^(-1/3)

  #fit NW estimators with box and Gaussian kernels
  y_box <- ksmooth(x, y, kernel = "box", x.points = x, bandwidth = h)$y

  y_gauss <- ksmooth(x, y, kernel = "normal", x.points = x, bandwidth = h)$y

  #calculate and format the MSEs
  mse <- colMeans( (cbind(y_1,y_2,y_3,y_4,y_5,y_box,y_gauss,
                         loess0_pred, loess1_pred,loess2_pred) - func(x))^2 )
  names(mse) <- c("Linear", paste("Poly Reg:", 2:5),
                "NW-Box","NW-Gaussian", paste("loess:", 0:2))

  return(mse)
}
```

Writing replicate function to prepare for simulation

```
replicate_func <- function(m,n,func){
  mse <- colMeans(matrix(replicate(m, simulate(n, func)), ncol = 10, byrow = TRUE))
  names(mse) <- c("Linear", paste("Poly Deg:", 2:5), "NW-Box","NW-Gaussian", paste("loess:", 0:2))
  return(mse)
}

#mse <-colMeans(matrix(replicate(m, simulate(1000, lin)), ncol = 10, byrow = TRUE))
m = 100
sample_size = c(100,500,1000,5000,10000)
```

First case

```
result1 = list()
for(i in 1:length(sample_size)){
  result1[[i]] = replicate_func(m,sample_size[i], lin)
}
result1
```

```
## [[1]]
##      Linear Poly Deg: 2 Poly Deg: 3 Poly Deg: 4 Poly Deg: 5      NW-Box
```

```
##  0.02091575  0.03256371  0.04286526  0.05432542  0.06584099  0.10193171
## NW-Gaussian    loess: 0    loess: 1    loess: 2
##  0.07884401  0.09015135  0.08966203  0.12068620
##
## [[2]]
##      Linear Poly Deg: 2 Poly Deg: 3 Poly Deg: 4 Poly Deg: 5      NW-Box
## 0.004473661 0.006352674 0.008267855 0.010235811 0.012112453 0.031844864
## NW-Gaussian    loess: 0    loess: 1    loess: 2
## 0.024771922 0.027296813 0.025120141 0.028593502
##
## [[3]]
##      Linear Poly Deg: 2 Poly Deg: 3 Poly Deg: 4 Poly Deg: 5      NW-Box
## 0.002259979 0.003046668 0.004166595 0.005160549 0.005982120 0.020598673
## NW-Gaussian    loess: 0    loess: 1    loess: 2
## 0.015746130 0.017242802 0.015345710 0.015901099
##
## [[4]]
##       Linear  Poly Deg: 2  Poly Deg: 3  Poly Deg: 4  Poly Deg: 5       NW-Box
## 0.0003594934 0.0005536060 0.0007305858 0.0009333253 0.0011030903 0.0068959040
##  NW-Gaussian     loess: 0     loess: 1     loess: 2
## 0.0052494828 0.0058262275 0.0050065543 0.0043774227
##
## [[5]]
##       Linear  Poly Deg: 2  Poly Deg: 3  Poly Deg: 4  Poly Deg: 5       NW-Box
## 0.0001762093 0.0002897800 0.0003999668 0.0004930913 0.0005908413 0.0042704842
##  NW-Gaussian     loess: 0     loess: 1     loess: 2
## 0.0032660614 0.0036948893 0.0031124091 0.0024901957
```

Second case

```r
result2 = list()
for(i in 1:length(sample_size)){
  result2[[i]] = replicate_func(m,sample_size[i], sin_2pi)
}
result2
```

```
## [[1]]
##      Linear Poly Deg: 2 Poly Deg: 3 Poly Deg: 4 Poly Deg: 5      NW-Box
##  0.21057075  0.21636547  0.04416759  0.05481001  0.06246595  0.10159219
## NW-Gaussian    loess: 0    loess: 1    loess: 2
##  0.07937388  0.09261970  0.08659256  0.11705202
##
## [[2]]
##      Linear Poly Deg: 2 Poly Deg: 3 Poly Deg: 4 Poly Deg: 5      NW-Box
##  0.19974264  0.20082929  0.01242095  0.01459214  0.01277658  0.03196768
## NW-Gaussian    loess: 0    loess: 1    loess: 2
##  0.02437023  0.02743659  0.02466988  0.02763701
##
## [[3]]
##      Linear Poly Deg: 2 Poly Deg: 3 Poly Deg: 4 Poly Deg: 5      NW-Box
## 0.196795328 0.197296037 0.008128362 0.008987163 0.005716019 0.019899531
## NW-Gaussian    loess: 0    loess: 1    loess: 2
## 0.015191138 0.016850526 0.014764347 0.015422621
##
```

```
## [[4]]
##      Linear Poly Deg: 2 Poly Deg: 3 Poly Deg: 4 Poly Deg: 5      NW-Box
## 0.196240711 0.196369557 0.005104318 0.005287240 0.001116486 0.006878702
## NW-Gaussian    loess: 0    loess: 1    loess: 2
## 0.005209786 0.005836905 0.004978541 0.004352118
##
## [[5]]
##       Linear  Poly Deg: 2  Poly Deg: 3  Poly Deg: 4  Poly Deg: 5       NW-Box
## 0.1961093158 0.1961573953 0.0047036131 0.0048087517 0.0005428125 0.0042608991
##  NW-Gaussian     loess: 0     loess: 1     loess: 2
## 0.0032353073 0.0036994920 0.0030801481 0.0024339360
```

Third case

```r
result3 = list()
for(i in 1:length(sample_size)){
  result3[[i]] = replicate_func(m,sample_size[i], sin_30)
}
result3
```

```
## [[1]]
##      Linear Poly Deg: 2 Poly Deg: 3 Poly Deg: 4 Poly Deg: 5      NW-Box
##   0.5094333   0.5127862   0.5049503   0.5111611   0.4911996   0.1799926
## NW-Gaussian    loess: 0    loess: 1    loess: 2
##   0.1955157   0.1881499   0.1950431   0.1615584
##
## [[2]]
##      Linear Poly Deg: 2 Poly Deg: 3 Poly Deg: 4 Poly Deg: 5      NW-Box
##  0.50090226  0.49941431  0.48409563  0.48474519  0.45786629  0.04549689
## NW-Gaussian    loess: 0    loess: 1    loess: 2
##  0.04950036  0.04885952  0.04944471  0.03860981
##
## [[3]]
##      Linear Poly Deg: 2 Poly Deg: 3 Poly Deg: 4 Poly Deg: 5      NW-Box
##  0.49661566  0.49525211  0.47922821  0.48008767  0.45325876  0.02875122
## NW-Gaussian    loess: 0    loess: 1    loess: 2
##  0.02963430  0.03099271  0.02877651  0.02284580
##
## [[4]]
##      Linear Poly Deg: 2 Poly Deg: 3 Poly Deg: 4 Poly Deg: 5      NW-Box
## 0.497083335 0.494966825 0.478522922 0.478590001 0.450516814 0.007795511
## NW-Gaussian    loess: 0    loess: 1    loess: 2
## 0.006981845 0.007838078 0.006498961 0.004868153
##
## [[5]]
##      Linear Poly Deg: 2 Poly Deg: 3 Poly Deg: 4 Poly Deg: 5      NW-Box
## 0.497080928 0.494876571 0.477993826 0.478021904 0.450404717 0.004696627
## NW-Gaussian    loess: 0    loess: 1    loess: 2
## 0.004000106 0.004615875 0.003709017 0.002691036
```

Tables

```r
res1 = data.frame(result1)
res2 = data.frame(result2)
```

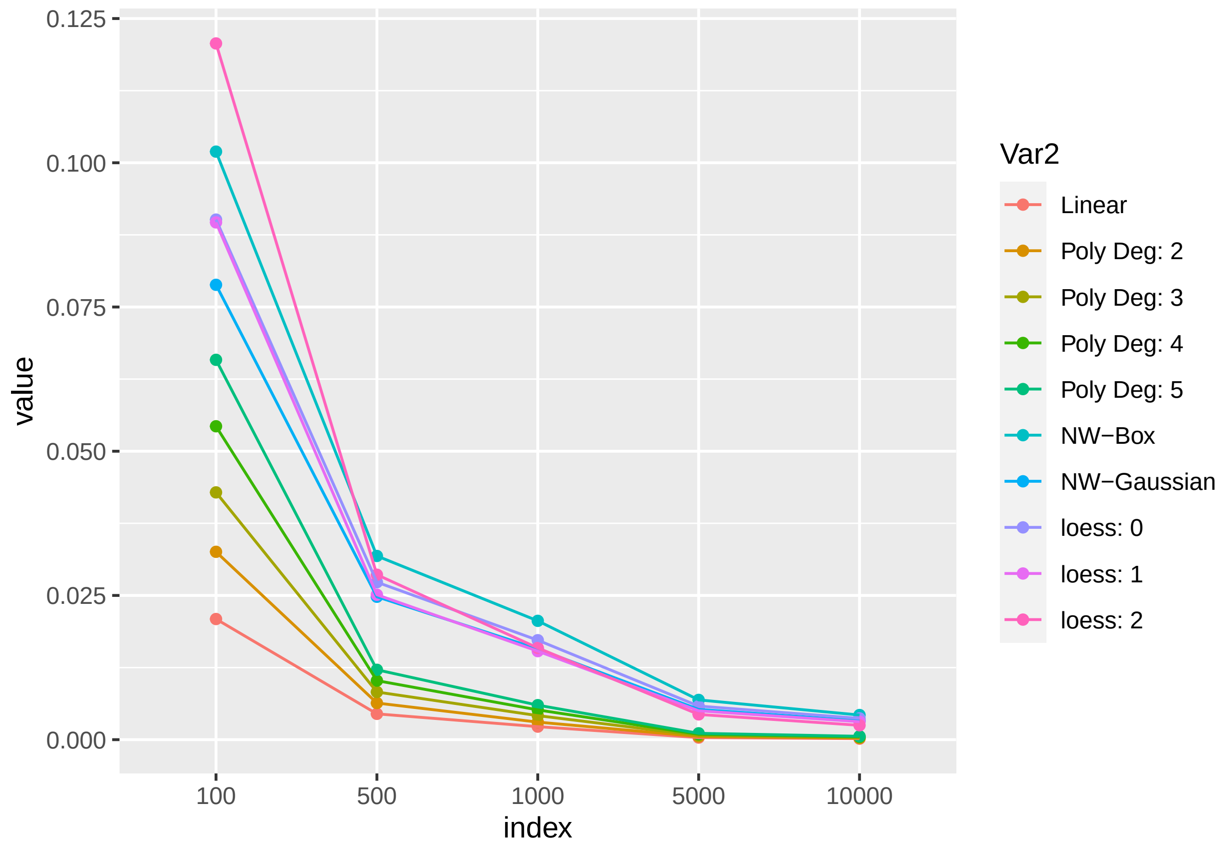
```
res3 = data.frame(result3)


names(res1) <- paste0("col", 1:5)
names(res2) <- paste0("col", 1:5)
names(res3) <- paste0("col", 1:5)

res1 = t(res1)
res2 = t(res2)
res3 = t(res3)
```

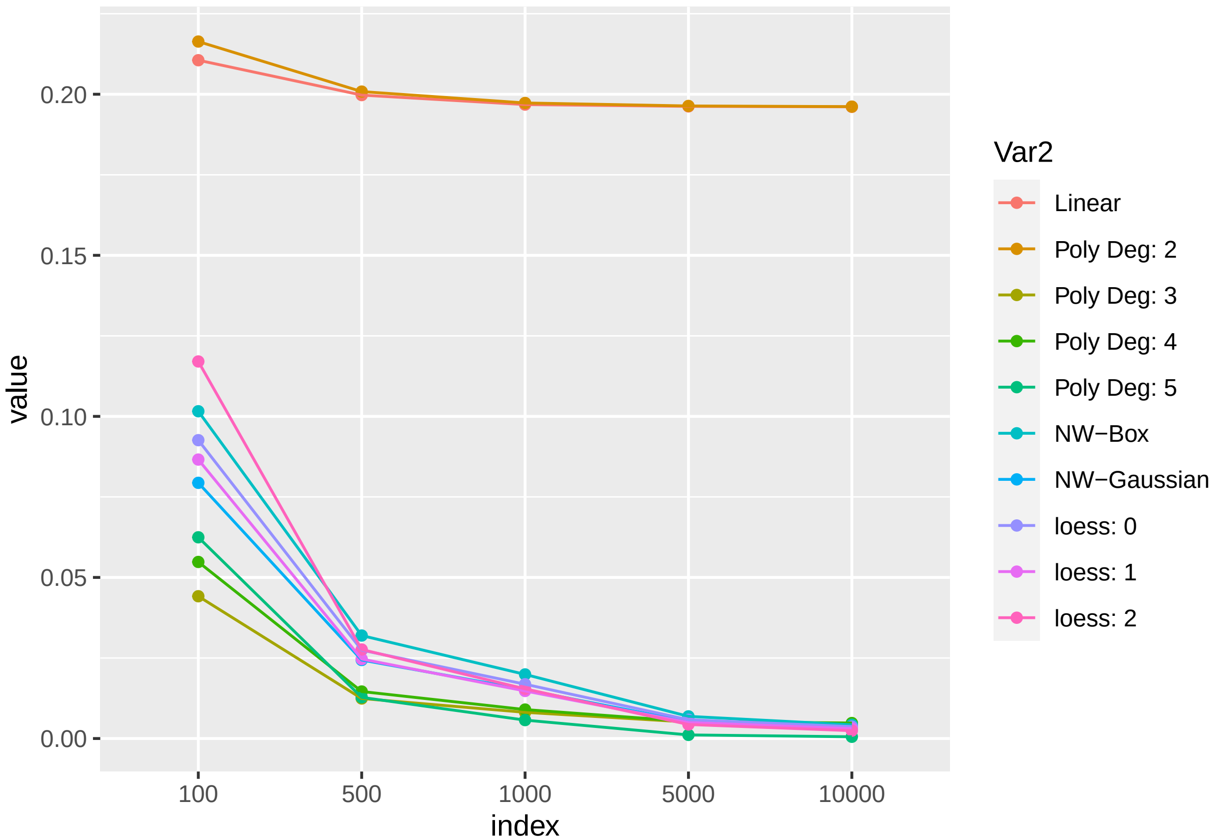
Plots

```
df1 = melt(res1)
df1$index = rep(as.factor(sample_size),10)
ggplot(df1,aes(x=index, y = value, color = Var2, group = Var2)) + geom_point() + geom_line()
```



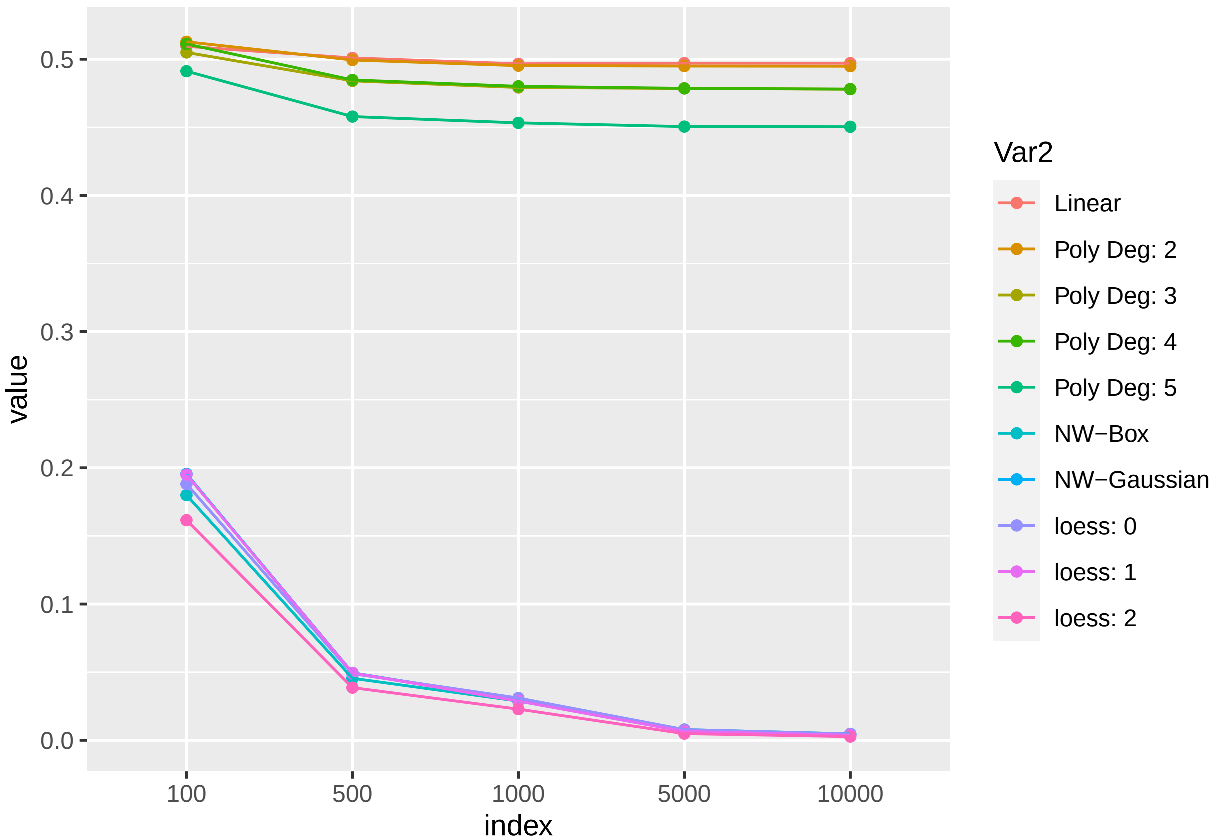
Linear

```
df2 = melt(res2)
df2$index = rep(as.factor(sample_size),10)
ggplot(df2,aes(x=index, y = value, color = Var2, group = Var2)) + geom_point() + geom_line()
```



sin 2 pi

```
df3 = melt(res3)
df3$index = rep(as.factor(sample_size),10)
ggplot(df3,aes(x=index, y = value, color = Var2, group = Var2)) + geom_point() + geom_line()
```



sin 30

Comments

The loess functions, compared with other methods, seem to perform well in general. And they perform especially well when the true function is most wiggly, potentially because the bandwidths were selected using the sin(30) data.

In general, the loess functions do not perform well if the sample size is small. One limitation is possibly that the R function LOESS only can go up to two degrees in calculating the curve. As sample size goes up, however, the loess seems to perform better than they do in smaller samples.

Based on our finite data, the LOESS outperforms the kernels as sample size increases. Note that loess with degree 0 is the NW estimator with a different bandwidth, and therefore they are pretty similar in performance.