ISYE 7406 Homework #4

Fall 2022

1. Introduction

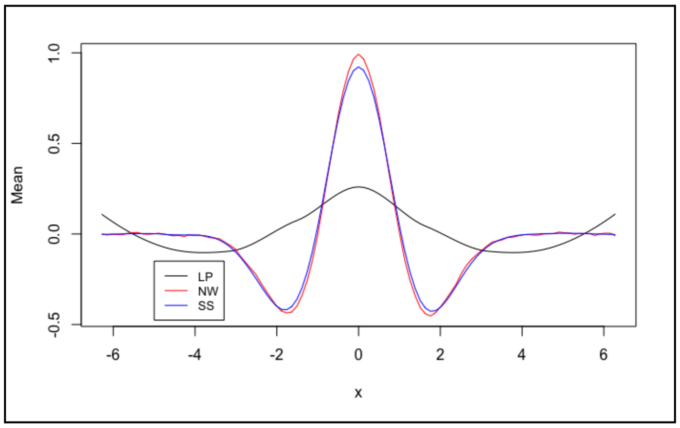
For his homework, we will be looking into three different local smoothing methods and their performance among a dataset generated by the “Mexican hat function”, and we’ll separate this analysis into two parts: equi-distant points among [-2π, 2π] and non-equidistant points also along [-2π, 2π]. The purpose of this is to evaluate their differences among statistical measures of mean, variance, bias, and mean squared error. We will also evaluate how performance changes with tuning parameters.

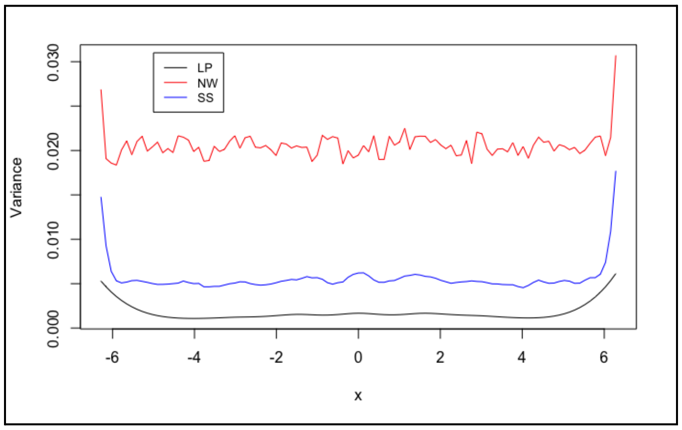
2. Experiment

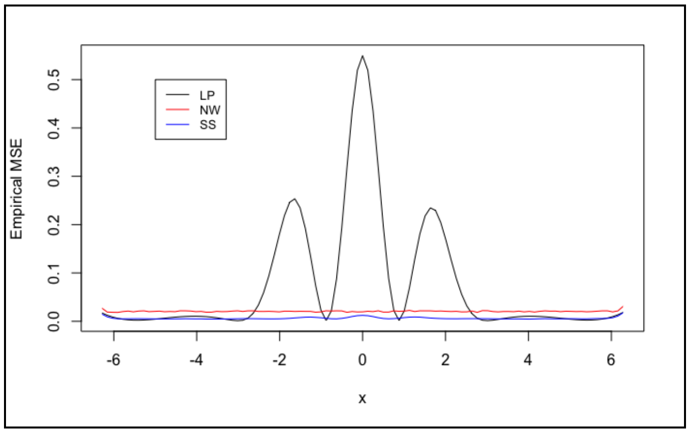
This experiment is separated into two parts with different datasets: equidistant points and non-equidistant points along [-2π, 2π]. For each dataset, we will compute m = 1000 Monte Carlo runs of each dataset where the dataset is randomly perturbed by an additive noise model. Then, we apply three different local smoothing methods: Loess (LP), Nadaraya-Watson (NW) smoothing, and spline smoothing. After 1000 runs, we aggregate our results to compare statistical measures.

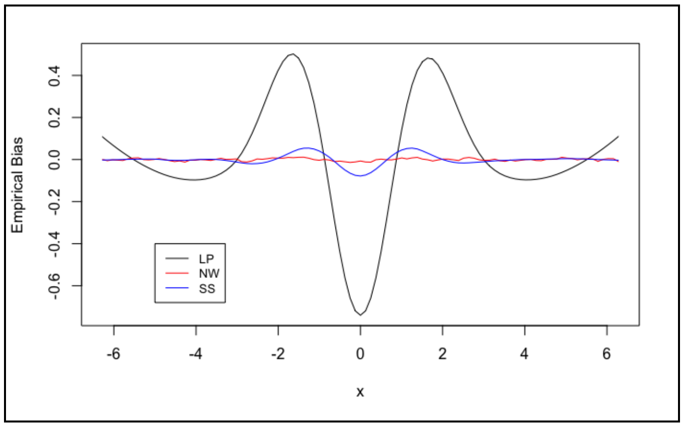
3. Part 1 Results

Below, we will plot 4 measures: mean, variance, bias, and mean squared error.



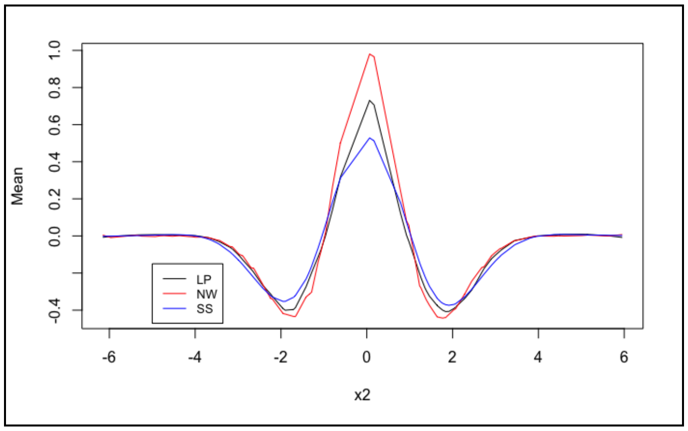


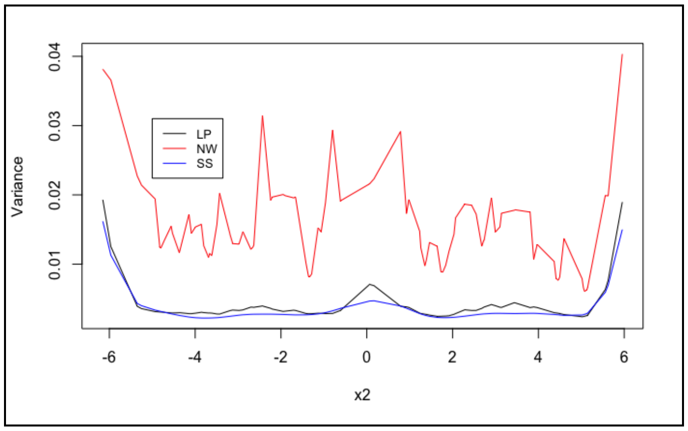


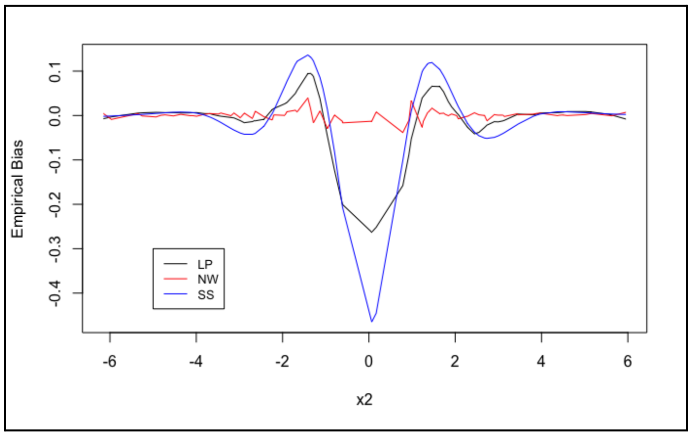


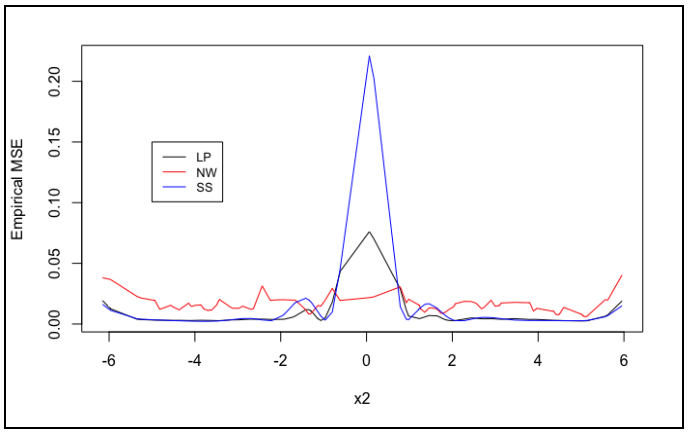
From our plots, we notice that the mean value for Loess smoothing shows a flatter line plot compared to the other two methods because the smoothing parameter (span = 0.75) is too high. Because of this relatively flat line, the Loess smoothed curve also has the least variance. In empirical mean-squared error and bias, we observe relatively low measures for NW and spline smoothing, but Loess is much higher in bias and MSE. Yet, the NW method and splint smoothing give similar results for mean, MSE, and bias, but NW provides the highest variance. This also further proves the relation MSE = Bias^2 + Variance since the Loess model has high bias but low variance suggesting this model under-fits the data with its high smoothing parameter. Overall, this is not a fair comparison between the three methods yet for which one best fits the data because we have not compared other hyperparameters. Let's take a look at the second dataset's results.

4. Part 2 Results









In Part 2, we are evaluating our smoothing models on a non-equidistant point dataset and also changing hyperparameter values. Loess span value is changed from 0.75 to 0.3365. Spline smoothing is now set with spar = 0.7163. NW bandwidth is kept at the same 0.2 value. We can see that now all three smoothing values look like they fit the dataset more closely compared to the previous one after changing hyperparameters. Loess and spline smoothing have similar results with low variance and higher bias compared to NW. They also have higher average mean-squared error with spline smoothing significantly worse at the center (x=0). This indicates that Loess and spline smoothing are underfitting in this dataset. Overall, the NW method appears to be the best smoothing model to fit this dataset. Its statistical measures are proof of this as well with its higher variance and low bias, relating to its low mean-squared error compared to the other two methods.

5. Findings

From our two datasets, it appeared that Nadaraya-Watson kernel smoothing fit both datasets quite well with low mean-squared error across the two compared to the other two smoothing methods. With data being more noisy in a real world-scenario as represented in Part 2, the NW method outperformed the other two making it a better model. However, this could change based on computer seed as well as evaluating different smoothing parameters.

Appendix: R Code

```
## Part #1 deterministic equidistant design
## Generate n=101 equidistant points in [-2\pi, 2\pi]
m <- 1000
n <- 101
x <- 2*pi*seq(-1, 1, length=n)

## Initialize the matrix of fitted values for three methods
fvlp <- fvnw <- fvss <- matrix(0, nrow= n, ncol= m)

##Generate data, fit the data and store the fitted values
for (j in 1:m){
## simulate y-values
## Note that you need to replace $f(x)$ below by the mathematical definition in
eq. (2)
fx <- (1-x^2) * exp(-0.5*x^2)
y <- fx + rnorm(length(x), sd=0.2);

## Get the estimates and store them
fvlp[,j] <- predict(loess(y ~ x, span = 0.75), newdata = x);
fvnw[,j] <- ksmooth(x, y, kernel="normal", bandwidth= 0.2, x.points=x)$y;
fvss[,j] <- predict(smooth.spline(y ~ x), x=x)$y
}

meanlp = apply(fvlp,1,mean);
meannw = apply(fvnw,1,mean);
meanss = apply(fvss,1,mean);

dmin = min( meanlp, meannw, meanss);
dmax = max( meanlp, meannw, meanss);
matplot(x, meanlp, "l", ylim=c(dmin, dmax), ylab="Mean")
matlines(x, meannw, col="red")
matlines(x, meanss, col="blue")
legend(-5, -0.15, legend=c("LP", "NW", "SS"),
       col=c("black", "red", "blue"), lty=1, cex=0.8)

var_lp = apply( (fvlp-meanlp)^2 , 1, sum)/m;
var_nw = apply( (fvnw-meannw)^2 , 1, sum)/m;
var_ss = apply( (fvss-meanss)^2 , 1, sum)/m;

dmin = min( var_lp, var_nw, var_ss);
dmax = max( var_lp, var_nw, var_ss);
matplot(x, var_lp, "l", ylim=c(dmin, dmax), ylab="Variance")
matlines(x, var_nw, col="red")
matlines(x, var_ss, col="blue")
legend(-5, 0.031, legend=c("LP", "NW", "SS"),
       col=c("black", "red", "blue"), lty=1, cex=0.8)

bias_lp = apply(fvlp,1,mean) - fx;
bias_nw = apply(fvnw,1,mean) - fx;
bias_ss = apply(fvss,1,mean) - fx;

dmin = min( bias_lp, bias_nw, bias_ss);
dmax = max( bias_lp, bias_nw, bias_ss);
matplot(x, bias_lp, "l", ylim=c(dmin, dmax), ylab="Empirical Bias")
matlines(x, bias_nw, col="red")
matlines(x, bias_ss, col="blue")
legend(-5, -0.4, legend=c("LP", "NW", "SS"),
       col=c("black", "red", "blue"), lty=1, cex=0.8)

mse_lp = apply( (fvlp-fx)^2 , 1, sum)/m;
mse_nw = apply( (fvnw-fx)^2 , 1, sum)/m;
mse_ss = apply( (fvss-fx)^2 , 1, sum)/m;

dmin = min( mse_lp, mse_nw, mse_ss);
dmax = max( mse_lp, mse_nw, mse_ss);
matplot(x, mse_lp, "l", ylim=c(dmin, dmax), ylab="Empirical MSE")
matlines(x, mse_nw, col="red")
matlines(x, mse_ss, col="blue")
legend(-5, 0.5, legend=c("LP", "NW", "SS"),
       col=c("black", "red", "blue"), lty=1, cex=0.8)

## Part 2
set.seed(79)
x2 <- round(2*pi*sort(c(0.5, -1 + rbeta(50,2,2), rbeta(50,2,2))), 8)
fvlp <- fvnw <- fvss <- matrix(0, nrow= n, ncol= m)

##Generate data, fit the data and store the fitted values
for (j in 1:m){
## simulate y-values
## Note that you need to replace $f(x)$ below by the mathematical definition in
eq. (2)
fx <- (1-x2^2) * exp(-0.5*x2^2)
y <- fx + rnorm(length(x2), sd=0.2);

## Get the estimates and store them
fvlp[,j] <- predict(loess(y ~ x2, span = 0.3365), newdata = x2);
fvnw[,j] <- ksmooth(x2, y, kernel="normal", bandwidth= 0.2, x.points=x2)$y;
fvss[,j] <- predict(smooth.spline(y ~ x2, spar=0.7163), x=x2)$y
}

meanlp = apply(fvlp,1,mean);
meannw = apply(fvnw,1,mean);
meanss = apply(fvss,1,mean);

dmin = min( meanlp, meannw, meanss);
dmax = max( meanlp, meannw, meanss);
matplot(x2, meanlp, "l", ylim=c(dmin, dmax), ylab="Mean")
matlines(x2, meannw, col="red")
matlines(x2, meanss, col="blue")
legend(-5, -0.15, legend=c("LP", "NW", "SS"),
       col=c("black", "red", "blue"), lty=1, cex=0.8)

var_lp = apply( (fvlp-meanlp)^2 , 1, sum)/m;
var_nw = apply( (fvnw-meannw)^2 , 1, sum)/m;
var_ss = apply( (fvss-meanss)^2 , 1, sum)/m;

dmin = min( var_lp, var_nw, var_ss);
dmax = max( var_lp, var_nw, var_ss);
matplot(x2, var_lp, "l", ylim=c(dmin, dmax), ylab="Variance")
matlines(x2, var_nw, col="red")
matlines(x2, var_ss, col="blue")
legend(-5, 0.031, legend=c("LP", "NW", "SS"),
       col=c("black", "red", "blue"), lty=1, cex=0.8)

bias_lp = apply(fvlp,1,mean) - fx;
bias_nw = apply(fvnw,1,mean) - fx;
bias_ss = apply(fvss,1,mean) - fx;

dmin = min( bias_lp, bias_nw, bias_ss);
dmax = max( bias_lp, bias_nw, bias_ss);
matplot(x2, bias_lp, "l", ylim=c(dmin, dmax), ylab="Empirical Bias")
matlines(x2, bias_nw, col="red")
matlines(x2, bias_ss, col="blue")
legend(-5, -0.3, legend=c("LP", "NW", "SS"),
       col=c("black", "red", "blue"), lty=1, cex=0.8)

mse_lp = apply( (fvlp-fx)^2 , 1, sum)/m;
mse_nw = apply( (fvnw-fx)^2 , 1, sum)/m;
mse_ss = apply( (fvss-fx)^2 , 1, sum)/m;

dmin = min( mse_lp, mse_nw, mse_ss);
dmax = max( mse_lp, mse_nw, mse_ss);
matplot(x2, mse_lp, "l", ylim=c(dmin, dmax), ylab="Empirical MSE")
matlines(x2, mse_nw, col="red")
matlines(x2, mse_ss, col="blue")
legend(-5, 0.15, legend=c("LP", "NW", "SS"),
       col=c("black", "red", "blue"), lty=1, cex=0.8)
```