Introduction

In this study, we delve into the evaluation of local smoothing methods, including LOESS, Nadaraya-Watson kernel smoothing, and spline smoothing. Our goal is to analyze their performance in estimating the Mexican hat function under different design scenarios. Specifically, we simulate datasets based on the additive noise model, considering equidistant and non-equidistant points, and assess the efficacy of these methods in capturing the underlying patterns. The challenges faced during these estimations, both statistical and computational, are thoroughly examined.

Exploratory Data Analysis

For our analysis, we simulate a dataset of (n = 101) observations using the additive noise model:

$$Y_i = f(x_i) + \epsilon_i$$

where *f(xi)* follows the Mexican hat function:

$$f(x) = (1 - x^2)\exp(-0.5x^2), \qquad -2\pi \leq x \leq 2\pi,$$

The error terms are independent and identically distributed with a mean of 0 and a standard deviation of 0.2.

Methods

LOESS (Local Polynomial Regression):

LOESS employs local smoothing to fit a polynomial surface determined by predictors. The choice of the span parameter significantly impacts flexibility, with higher spans potentially causing over-smoothing.

Nadaraya-Watson Kernel Smoothing:

Nadaraya-Watson estimates the regression function through weighted averages using a kernel function. The bandwidth parameter is crucial, striking a balance between bias and variance.

Spline Smoothing

Spline smoothing utilizes cubic smoothing splines with a smoothness penalty. The tuning parameter controls the tradeoff between fitting the data and maintaining smoothness.

Results

Equidistant Design

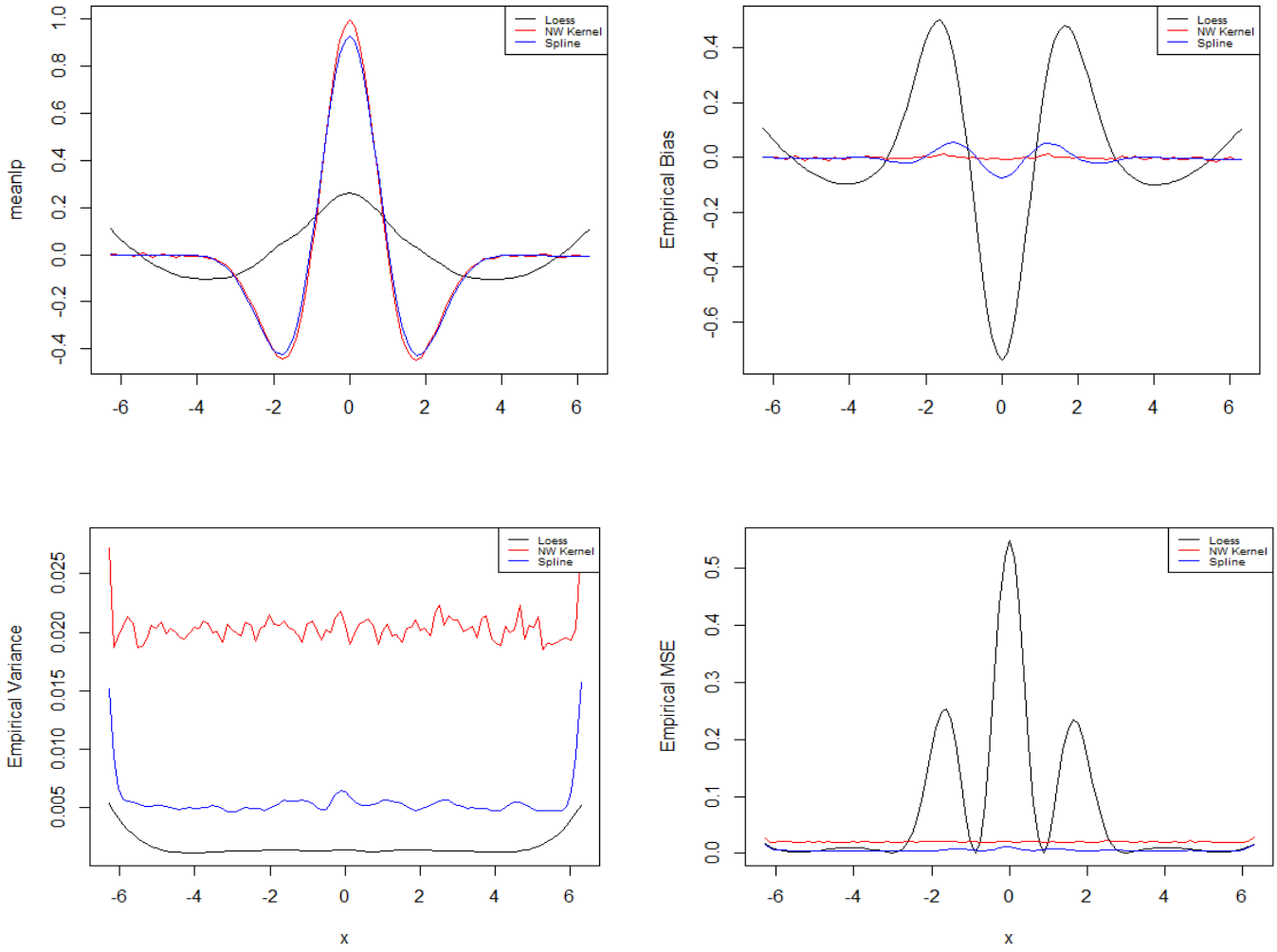

Figure 1 : The plots from top left to bottom right: fitted mean curves, empirical bias, empirical variance, and MSE for the 3 methods:

Non-Equidistant Design:

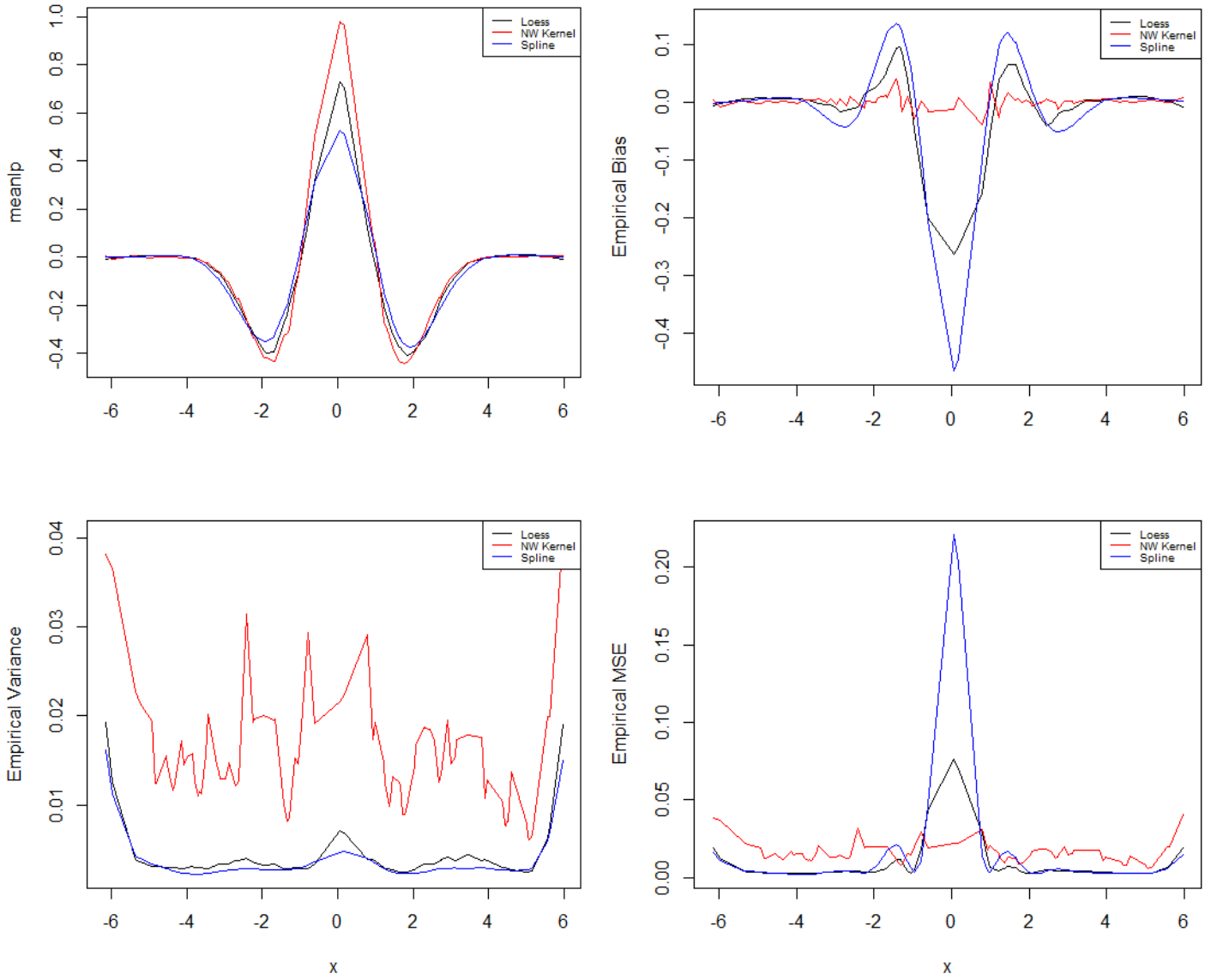

Figure 2 :Empirical Plot for Non-equidistant. The plots from top left to bottom right: fitted mean curves, empirical bias, empirical variance, and MSE for three methods:

Findings and Conclusion

From our analysis, we observe that the choice of tuning parameters significantly influences the performance of local smoothing methods. In the equidistant design, LOESS tends to over smooth

the data when the span parameter is high, leading to higher bias. Nadaraya-Watson and spline smoothing show more resilience, capturing the underlying patterns more accurately.

In the non-equidistant design, tuning parameters play a critical role in mitigating bias and variance. LOESS, with a lower span, fits the fluctuating data better, resulting in reduced bias. Nadaraya-Watson, with an appropriately chosen bandwidth, demonstrates exceptional performance in this scenario.

Challenges and Considerations

Statistical Challenges:

1. Variability and Bias Around x=0:

The empirical bias, variance, and mean square errors exhibit substantial fluctuations, particularly around x=0. This observation signifies a significant statistical challenge, indicating the difficulty in accurately estimating the Mexican hat function, especially in the vicinity of x=0. The erratic behavior of these metrics at this point suggests inherent complexities in capturing the true function in this region.

2. Bias-Variance Tradeoff: Striking the right balance between bias and variance is challenging. Adjusting parameters to minimize mean square error while maintaining interpretability is vital.

Computational Challenges:

1. Computational Intensity: LOESS computations can be intensive for large datasets, and optimizing parameters adds complexity. Nadaraya-Watson involves substantial calculations due to the weighted averages. Spline smoothing's iterative optimization can be time-consuming.

2. Tuning Parameter Selection: Tuning parameter optimization involves exploring the tradeoffs. Automated methods can be computationally expensive but are necessary for robust results.

Project 4 appendix

Ron Mosness

2023-10-03

```
# Generate n=101 equidistant points
m <- 1000
n <- 101
x <- 2*pi*seq(-1, 1, length=n)

# Initialize the matrix of fitted values
fvlp <- fvnw <- fvss <- matrix(0, nrow= n, ncol= m)
#Generate data fit the data and store
for (j in 1:m){

  # simulate y-values
  y <- (1-x^2) * exp(-0.5 * x^2) + rnorm(length(x), mean =0, sd = 0.2);
  # estimates finded
  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;
}

# mean estimates
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))
matlines(x, meannw, col="red")
matlines(x, meanss, col="blue")
legend("topright", legend=c("Loess", "NW Kernel", "Spline"), col=c("black", "
red", "blue"), lty=1, cex=0.6)

# Plot empirical Bias
ytrue = (1-x^2) * exp(-0.5 * x^2);
biaslp = apply(fvlp,1,mean) - ytrue;
biasnw = apply(fvnw,1, mean) - ytrue;
biasss = apply(fvss,1, mean) - ytrue;
d1min = min(biaslp, biasnw, biasss);
d1max = max(biaslp, biasnw, biasss);
matplot(x, biaslp, "l", ylim=c(d1min, d1max), ylab="Empirical Bias");
matlines(x, biasnw, col="red");
matlines(x, biasss, col="blue");
legend("topright", legend=c("Loess", "NW Kernel", "Spline"), col=c("black", "
red", "blue"), lty=1, cex=0.6)
```

```
# Plot empirical Variance
Varlp = apply( (fvlp - meanlp)^2, 1,sum) / m;
Varnw = apply( (fvnw - meannw)^2, 1,sum) / m;
Varss = apply( (fvss - meanss)^2, 1,sum) / m;
d2min = min(Varlp, Varnw, Varss);
d2max = max(Varlp, Varnw, Varss);
matplot(x, Varlp, "l", ylim=c(d2min, d2max), ylab="Empirical Variance");
matlines(x, Varnw, col="red");
matlines(x, Varss, col="blue");
legend("topright", legend=c("Loess", "NW Kernel", "Spline"), col=c("black", "
red", "blue"), lty=1, cex=0.6)

# Plot empirical MSE
MSElp = apply( (fvlp - ytrue)^2, 1,sum) / m;
MSEnw = apply( (fvnw - ytrue)^2, 1,sum) / m;
MSEss = apply( (fvss - ytrue)^2, 1,sum) / m;
d3min = min(MSElp, MSEnw, MSEss);
d3max = max(MSElp, MSEnw, MSEss);
matplot(x, MSElp, "l", ylim=c(d3min, d3max), ylab="Empirical MSE");
matlines(x, MSEnw, col="red");
matlines(x, MSEss, col="blue");
legend("topright", legend=c("Loess", "NW Kernel", "Spline"), col=c("black", "
red", "blue"), lty=1, cex=0.6)

## Non-equidistant points
set.seed(79)
x <- 2*pi*sort(c(0.5, -1 + rbeta(50,2,2), rbeta(50,2,2)))
fvlp <- fvnw <- fvss <- matrix(0, nrow= n, ncol= m)
#Generate and fit data
for (j in 1:m){
  # simulate y-values
  y <- (1-x^2) * exp(-0.5 * x^2) + rnorm(length(x), sd=0.2);
  fvlp[,j] <- predict(loess(y ~ x, span = 0.3365), newdata = x);
  fvnw[,j] <- ksmooth(x, y, kernel="normal", bandwidth= 0.2, x.points=x)$y;
  fvss[,j] <- predict(smooth.spline(y ~ x, spar= 0.7163), x=x)$y;
}
# mean estimator
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))
matlines(x, meannw, col="red")
matlines(x, meanss, col="blue")
legend("topright", legend=c("Loess", "NW Kernel", "Spline"), col=c("black", "
red", "blue"), lty=1, cex=0.6)

# Plot empirical Bias
ytrue = (1-x^2) * exp(-0.5 * x^2);
```

```
biaslp = apply(fvlp,1,mean) - ytrue;
biasnw = apply(fvnw,1, mean) - ytrue;
biasss = apply(fvss,1, mean) - ytrue;
d1min = min(biaslp, biasnw, biasss);
d1max = max(biaslp, biasnw, biasss);
matplot(x, biaslp, "l", ylim=c(d1min, d1max), ylab="Empirical Bias");
matlines(x, biasnw, col="red");
matlines(x, biasss, col="blue");
legend("topright", legend=c("Loess", "NW Kernel", "Spline"), col=c("black", "
red", "blue"), lty=1, cex=0.6)

# Plot empiricall Variance
Varlp = apply( (fvlp - meanlp)^2, 1,sum) / m;
Varnw = apply( (fvnw - meannw)^2, 1,sum) / m;
Varss = apply( (fvss - meanss)^2, 1,sum) / m;
d2min = min(Varlp, Varnw, Varss);
d2max = max(Varlp, Varnw, Varss);
matplot(x, Varlp, "l", ylim=c(d2min, d2max), ylab="Empirical Variance");
matlines(x, Varnw, col="red");
matlines(x, Varss, col="blue");
legend("topright", legend=c("Loess", "NW Kernel", "Spline"), col=c("black", "
red", "blue"), lty=1, cex=0.6)

# Plot empirical MSE
MSElp = apply( (fvlp - ytrue)^2, 1,sum) / m;
MSEnw = apply( (fvnw - ytrue)^2, 1,sum) / m;
MSEss = apply( (fvss - ytrue)^2, 1,sum) / m;
d3min = min(MSElp, MSEnw, MSEss);
d3max = max(MSElp, MSEnw, MSEss);
matplot(x, MSElp, "l", ylim=c(d3min, d3max), ylab="Empirical MSE");
matlines(x, MSEnw, col="red");
matlines(x, MSEss, col="blue");
legend("topright", legend=c("Loess", "NW Kernel", "Spline"), col=c("black", "
red", "blue"), lty=1, cex=0.6)
```