**Introduction**

This homework's goal is to get familiar with local smoothing. There are multiple kinds of smoothing, and this assignment focuses on 3 of them – Locally estimated scatterplot smoothing (LOESS), Nadaraya-Wastson (NW) kernel smoothing, and spline smoothing. Each smoothing method had its empirical variance, empirical variance, and empirical mean square error looked at and compared.

Two different data sets were used for this assignment, although they are similar. Both data sets are a simulation of 101 observations from the additive noise model $Y_i = f(x_i) + \mathcal{E}_i$ where the function is the Mexican hat function $f(x) = (1 - x^2)\exp(-0.5x^2), -2\pi \leq x \leq 2\pi$. Where they differ is the first data set has equidistant points in $[-2\pi, 2\pi]$ and the second data set does not.

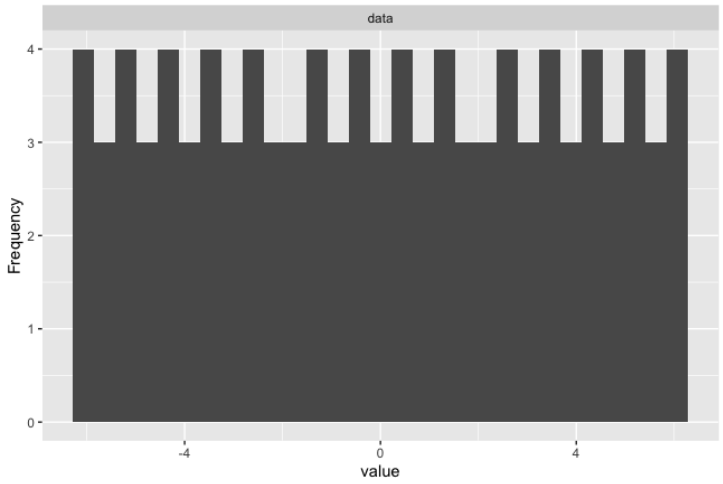
**Exploratory Data Analysis**

Equidistant data set:

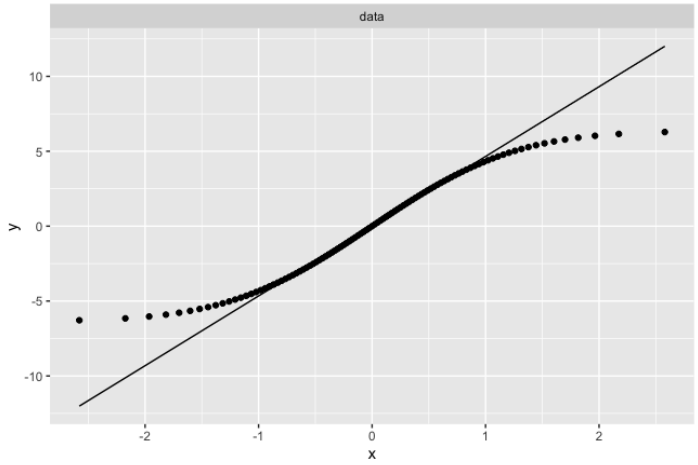
The equidistant data set is comprised of 1 column with 101 rows. Below is some information about the values of the column:

| Min | 1st Quartile | Median | Mean | 3rd Quartile | Max |
|---|---|---|---|---|---|
| -6.283 | -3.142 | 0.000 | 0.000 | 3.142 | 6.238 |

Due to this data set being equidistant points in $[-2\pi, 2\pi]$, these are the values that were expected to be seen. To confirm this information, a histogram was looked at to see the spread of the data set:




As the information in the table suggested, the data is split evenly along the mean of 0. There are the same exact negative values that match the same positive values, which would result in a mean of 0. By looking at the histogram, the data set does not follow a normal distribution. To look at how this data set compares to a normal distribution, a qq plot was looked at:



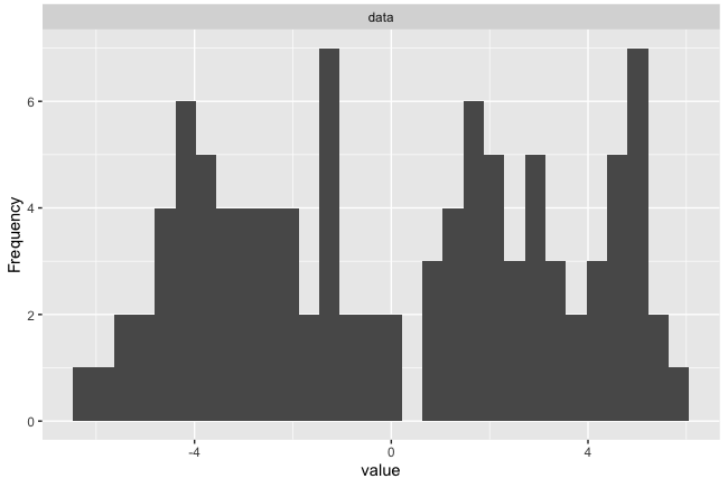

The points on the plot show a S-shaped curve instead of a straight one, which indicates that the data is not normally distributed. This suggests that the data set follows a nonlinear function, which it does not follow because $x_i$ is defined as $x_i = 2\pi\ (-1 + 2\left(\frac{i-1}{n-1}\right))$ for this data set, which in nonlinear. By looking at the summary information, a histogram, and a qq plot, it is shown that this data set is doing what it is assumed to be doing.

Non-equidistant data set:

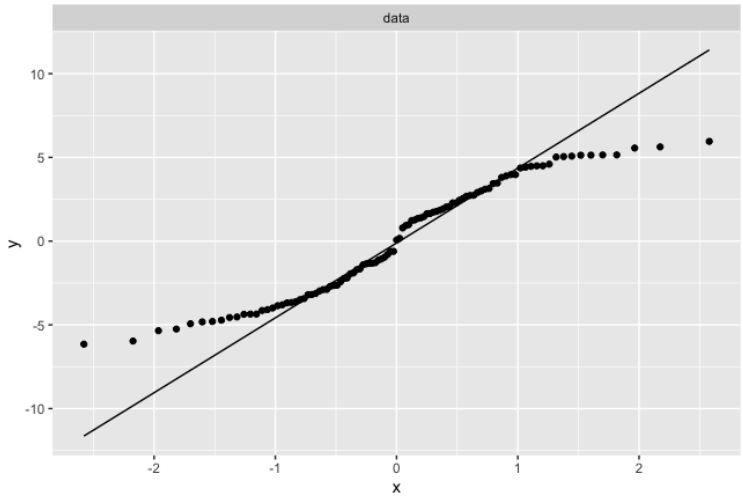
Like the data set above, this data set is also comprised of 1 column with 101 rows. Below is the information about this column:

| Min | 1st Quartile | Median | Mean | 3rd Quartile | Max |
|---|---|---|---|---|---|
| -6.153 | -3.125 | 0.066 | 0.022 | 2.905 | 5.955 |

Looking at this table, it is already apparent that this data set is different than the first one. Unlike the first one, this data set's min and max are not $-2\pi$ and $2\pi$, its 1st and 3rd quartiles are not the same positive and negative values of each other, and the median and mean are not 0. This all indicates that the values to the left and right of 0 are not the same. A mean of 0.022 shows that there are more positive values than negative. To explore the spread of this data set more, a histogram was looked at:



This histogram shows that this data set is far from normal. It looks like the histogram could be broken down into two separate ones from how the data is distributed. Looking closer at the data, this data set does not have a data point between -0.61076290 and 0.06614811 which explains the missing bar in the middle of the figure. To see how this data set compares to a normal distribution, a qq plot was ran:




Unlike the first data set which had multiple data points follow the diagonal line, this data set barely has any that do. This indicates that this data set is even less normal than the first data set. This makes sense because this data set was created with a similar $x_i$ as the first data set, but this one did not follow ( $-1 + 2\left(\frac{i-1}{n-1}\right)$) that the first one did. Like the first data set, looking at this table, histogram, and qq plot, it is shown that the data set is doing what it is assumed to be doing.

**Methods/Methodology**

There were three methods used in this assignment. All three of them were a form of local smoothing.

The first model used was locally estimated scatterplot smoothing (LOESS). LOESS is used to fit a smooth curve to a set of data points, especially when the data has a lot of noise. It fits local models to small subsets of data. The way it works is that for each data point, it looks at a neighborhood around that data point based on a specific span and fits a simple model (usually linear or quadratic regression) to the points in that neighborhood. After the local models are fit, LOESS combines the local fits into a smooth curve that represents the trend in the data as a whole.

The second model used was Nadaraya-Watson (NW) kernel smoothing. NW kernel smoothing is a type of kernel regression, meaning it uses kernel functions to estimate the value of the dependent variable for a given input based on its local neighborhood. The kernel function determines how much weight each data points has based on its distance from the target point. Like LOESS, it also has a bandwidth/span value for the size of the neighborhood it looks at.

The last model used was spline smoothing. Spline smoothing allows for a piecewise smooth function that can capture complex patterns in data more accurately. It fits a smooth curve through the data points using piecewise polynomials (splines) which are connected at certain points called knots. Its purpose is to combine polynomials that result in a smooth and flexible curve. There are different kinds of spline smoothing like cubic, B, and natural to name a few.

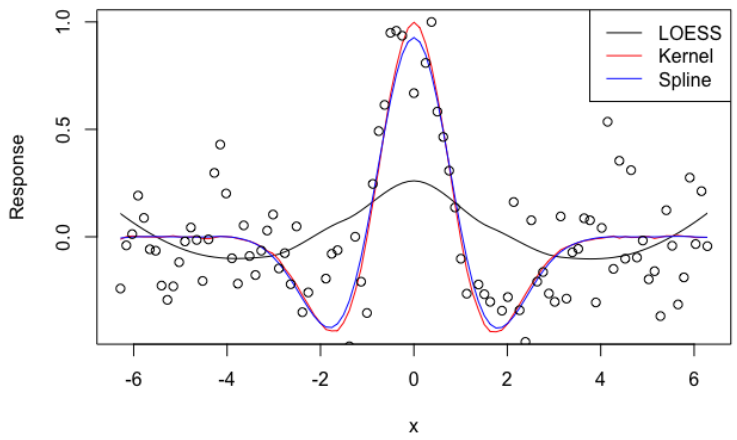
Each local smoothing model was run with m = 1000 Monte Carlo simulations. Monte Carlo simulations are a way to model and understand systems that involve uncertainty or randomness. They use repeated random sampling to simulate possible outcomes, in this case helping to estimate the results of each model.

For each local smoothing method, based on m = 1000 Monte Carlo runs, the empirical bias, empirical variance, and empirical mean square error were computed and plotted against $x_i$. The relation $MSE = Bias^2 + Var$ is applicable for these values.

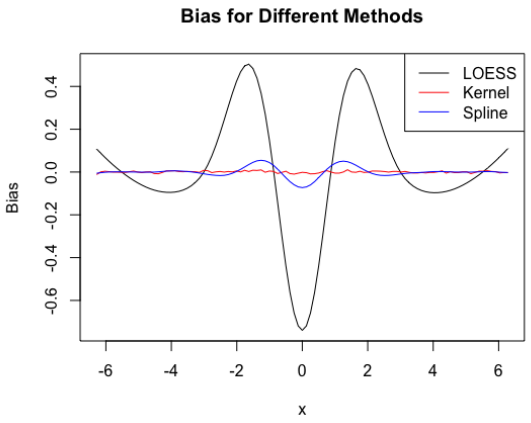
**Results**

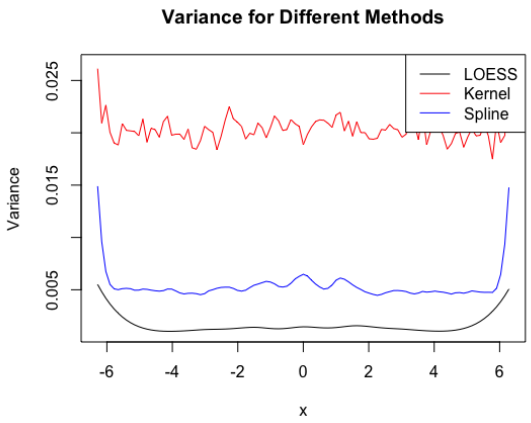
Equidistant data set:

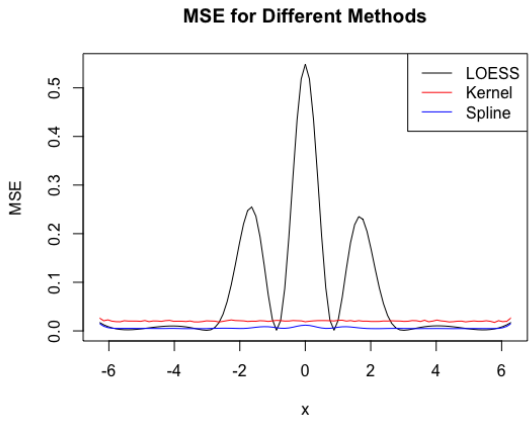
For this data set, LOESS was used with a span of 0.75, NW kernel smoothing was used with the Gaussian Kernel and a bandwidth of 0.2, and spline smoothing was used with the default tuning parameter. Below is a plot of the mean of the three estimators:




This graph shows that the LOESS fit is much smoother than that of NW and spline but is also less responsive to the fluctuations in the data. It does not show the sharp peak at x = 0 like the other two do, meaning it underfits that region. NW fits the sharp beat at x = 0 and captures the general trend of the data. Spline's fit is like NW but is slightly smoother. Below are the graphs showing the bias, variance, and mean square error for each model:



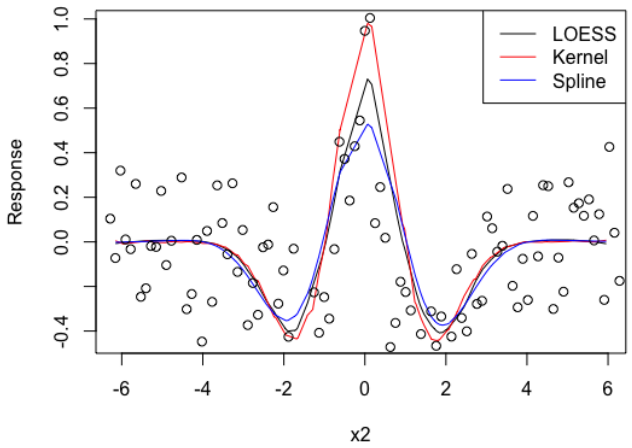

Also below is the average bias, variance and mean square error for each model:

| | Bias | Variance | Mean Square Error |
|---|---|---|---|
| LOESS | 0.0145992766 | 0.001638227 | 0.076792639 |
| NW Kernel Smoothing | 0.0002321442 | 0.020340432 | 0.020359211 |
| Spline Smoothing | 0.0002259430 | 0.005398896 | 0.005986517 |

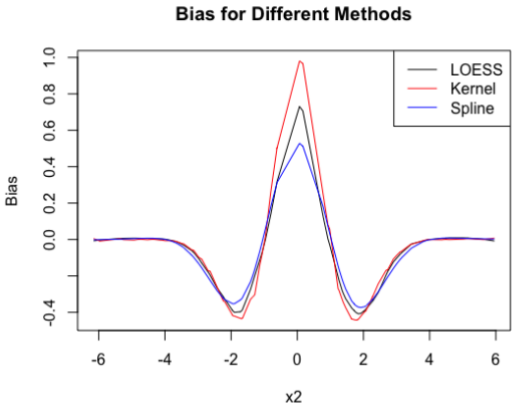
LOESS having the largest average bias shows when looking at the graphs. Due to the model not being as responsive as the other two to the fluctuations around x = 0, the bias graph shows a large negative value for that region – which means the model's predictions are lower than the true values, which they are. NW and spline both capture the general trend of the data, which their bias values and trends reflect. Although it had the highest bias, LOWESS had the lowest variance. This is largely due to its local fitting and smoothing. Localized fitting makes it less sensitive to data points outside of the span it looks at, which keeps variance low. Spline seemed to have high variance at the min and max points and had a relatively smooth variance line in-between those values. NW had much larger variance numbers than the two. LOWESS had the largest mean square error values. Looking at the graphs, it is shown that it happens from the -2 to 2 range, which is where the model underfit the data points. NW and spline followed a similar trend in their mean squared errors, but spline produced lower values than NW. This is likely to spline performing global smoothing (fits a smooth curve over all the data) where the other two do local smoothing. If the data itself has a smooth trend, which this one does, spline will likely approximate better.

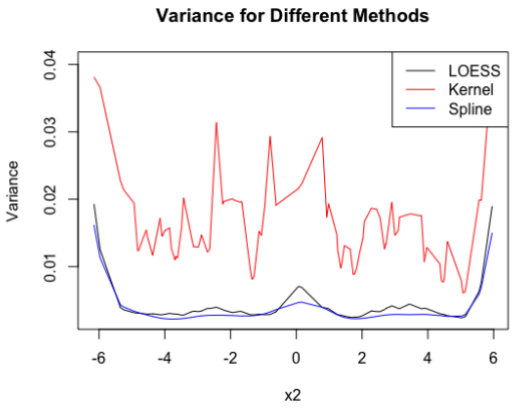
Non-equidistant data set:

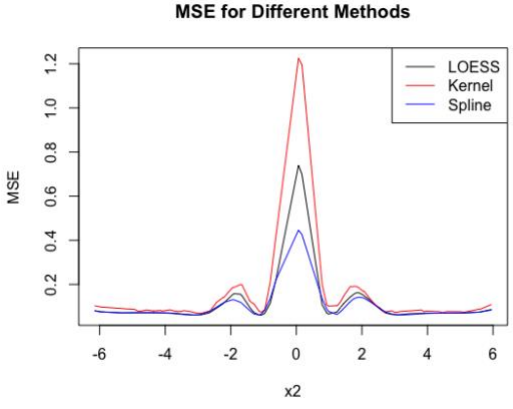
For this data set, LOESS was used with a span of 0.3365, NW kernel smoothing was used with the Gaussian Kernel and a bandwidth of 0.2, and spline smoothing was used with the tuning parameter spar = 0.7163. Below is a plot of the mean of the three estimators:




When looking at this graph, it is apparent that these models have a harder time smoothing out the data than they did with the equidistant data set. Each model follows the same general trend, the main difference is that each one fits the data points around x2 = 0. In this data set, NW looks to be overfitting the data set by reacting to the two data points at the top more than the other two models. Below are the graphs showing the bias, variance, and mean square error for each model:



Also below is the average bias, variance and mean square error for each model:

| | Bias | Variance | Mean Square Error |
|---|---|---|---|
| LOESS | -0.09155372 | 0.003728249 | 0.10090093 |
| NW Kernel Smoothing | -0.08793551 | 0.015194674 | 0.13159221 |
| Spline Smoothing | -0.08783299 | 0.003211542 | 0.09209863 |

Based on this table, LOESS shows the highest negative bias on average, meaning that it underfits the data the most of the three. Even though in the plot of the three models it looked like NW was overfitting the data set, it underfits the data set on average just as much as spline did. Spline and NW having around the same bias values mean they provide a more accurate estimation than LOESS does. Like the first data set, LOESS provided very low values for variance, meaning that the model was consistent in both tests. Also, like the first data set, NW had the largest variance values by far. What was different between the two is that in this case spline had the lowest variance value, not LOESS. This is likely due to spline's global fitting which doesn't rely on neighborhoods and the added penalty term that is meant to discourage overfitting which would reduce the variance. Due to $MSE = Bias^2 + Var$, it makes sense for NW to have the largest mean square error and spline to have the smallest because of their bias and variance values.

**Findings**

For both data sets, if I were to pick a model to use, I would use spline smoothing for both. Spline resulted in the lowest bias for both, a good variance for both, and the lowest mean square error for both. This model provided the best tradeoff between bias and variance for both data sets. The Mexican hat function has sharp peaks and oscillations in it – NW kernel smoothing lead to errors in trying to over smooth those peaks and LOESS tried to under smooth in these data sets.

I found a couple things interesting about these results. In terms of variance, I found it odd that NW kernel smoothing produced a much more volatile looking chart than LOESS and spline did. After looking into it, I saw that a smaller bandwidth can lead to more volatility because it makes the local fits tighter than LOESS and spline would. Another thing I found interesting was the LOESS bias and mean square error graphs for the first data set. I wasn't necessarily surprised that the bias was much higher than the other two because it did underfit the data much more than the others, I was more interested at the fact that the mean squared error graph was pretty much the bias graph flipped 180 degrees. I looked back at the homework prompt and saw the equation $MSE = Bias^2 + Var$, which made it all make sense.

In terms of the non-equidistant data set, each model had both statistical and computational challenges. Some statistical challenges the smoothing methods that use neighborhoods (NW and LOESS) face are that they generally assume uniform data distribution, the Mexican hat function has a sharp peak and oscillations which LOESS and NW struggle with rapid changes – they generally over smooth peaks and under smooth oscillations, and their results can be made worse when there are fewer nearby points in the span/bandwidth. Spline's statistical challenges were that if there is a region where there are few data points it will create artificial oscillations, it can create excessive

curves at boundaries, and it usually does not do good handling sharp jumps in the data. The computational challenge of LOESS is its computational cost. It relies on fitting small curves to different parts of the data, which makes it computationally expensive, especially for large data sets. The challenge of NW kernel smoothing is that it needs careful tuning. It relies on averaging nearby points, but picking the right bandwidth is difficult. In a real-world setting, multiple different options need to be tested, which add extra computation time. It also struggles with high dimensional data. Spline smoothing's computational challenges come from choosing the right number and placement of knots. If there are too few knots, the spline may not include important patterns in the data, and if there are too many then it may be overly sensitive to small changes in the data. Finding the right number and placement often requires testing many different ones or an optimization problem.

**Appendix**

```
#PART 1 _____

set.seed(123)
library(DataExplorer)

## True function (f(x)) definition (eq. (2) as given)
f_true <- function(x) (1 - x^2) * exp(-0.5 * x^2)

## Number of simulations (Monte Carlo runs)
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)


length(x)
summary(x)
plot_histogram(x)
plot_qq(x)


## Run the simulations
for (j in 1:m) {
  ## Simulate y-values based on true function and noise
```

```
  y <- (1 - x^2) * exp(-0.5 * x^2) + 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
}

## Initialize vectors to store the empirical bias, variance, and MSE for each method
bias_lp <- variance_lp <- mse_lp <- numeric(n)
bias_nw <- variance_nw <- mse_nw <- numeric(n)
bias_ss <- variance_ss <- mse_ss <- numeric(n)

## Compute bias, variance, and MSE for each method at each x_i
for (i in 1:n) {
  ## Collect the fitted values for the current x_i across all simulations
  y_lp <- fvlp[i, ]
  y_nw <- fvnw[i, ]
  y_ss <- fvss[i, ]

  ## True value at x_i
  true_value <- f_true(x[i])

  ## Compute the mean of the fitted values
  mean_lp <- mean(y_lp)
  mean_nw <- mean(y_nw)
  mean_ss <- mean(y_ss)
```

```
  ## Compute Bias (average of estimates - true value)
  bias_lp[i] <- mean_lp - true_value
  bias_nw[i] <- mean_nw - true_value
  bias_ss[i] <- mean_ss - true_value

  ## Compute Variance (variance of estimates)
  variance_lp[i] <- mean((y_lp - mean_lp)^2)
  variance_nw[i] <- mean((y_nw - mean_nw)^2)
  variance_ss[i] <- mean((y_ss - mean_ss)^2)

  ## Compute MSE (mean squared error between estimates and true value)
  mse_lp[i] <- mean((y_lp - true_value)^2)
  mse_nw[i] <- mean((y_nw - true_value)^2)
  mse_ss[i] <- mean((y_ss - true_value)^2)
}

## Below is the sample R code to plot the mean of three estimators in a single plot
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="Response")
matlines(x, meannw, col="red")
matlines(x, meanss, col="blue")
legend("topright", legend=c("LOESS", "Kernel", "Spline"), col=c("black", "red", "blue"), lty=1)
```

```
## You might add the raw observations to compare with the fitted curves
points(x,y)

## Plot the empirical bias, variance, and MSE

par(mar=c(5, 5, 4, 6))

## Plot Bias
plot(x, bias_lp, type="l", col="black", ylim=c(min(bias_lp, bias_nw, bias_ss), max(bias_lp, bias_nw, bias_ss)),
    ylab="Bias", xlab="x", main="Bias for Different Methods")
lines(x, bias_nw, col="red")
lines(x, bias_ss, col="blue")
legend("topright", legend=c("LOESS", "Kernel", "Spline"), col=c("black", "red", "blue"), lty=1)

## Plot Variance
plot(x, variance_lp, type="l", col="black", ylim=c(min(variance_lp, variance_nw, variance_ss), max(variance_lp, variance_nw, variance_ss)),
    ylab="Variance", xlab="x", main="Variance for Different Methods")
lines(x, variance_nw, col="red")
lines(x, variance_ss, col="blue")
legend("topright", legend=c("LOESS", "Kernel", "Spline"), col=c("black", "red", "blue"), lty=1)

## Plot MSE
plot(x, mse_lp, type="l", col="black", ylim=c(min(mse_lp, mse_nw, mse_ss), max(mse_lp, mse_nw, mse_ss)),
    ylab="MSE", xlab="x", main="MSE for Different Methods")
lines(x, mse_nw, col="red")
```

```
lines(x, mse_ss, col="blue")
legend("topright", legend=c("LOESS", "Kernel", "Spline"), col=c("black", "red", "blue"), lty=1)

# Now compute summary statistics for each method
bias_lp_mean <- mean(bias_lp)
var_lp_mean <- mean(variance_lp)
mse_lp_mean <- mean(mse_lp)

bias_nw_mean <- mean(bias_nw)
var_nw_mean <- mean(variance_nw)
mse_nw_mean <- mean(mse_nw)

bias_ss_mean <- mean(bias_ss)
var_ss_mean <- mean(variance_ss)
mse_ss_mean <- mean(mse_ss)

# Create a data frame to display the results in a readable format
summary_stats <- data.frame(
  Method = c("LOESS", "Kernel Smoothing", "Spline Smoothing"),
  Bias = c(bias_lp_mean, bias_nw_mean, bias_ss_mean),
  Variance = c(var_lp_mean, var_nw_mean, var_ss_mean),
  MSE = c(mse_lp_mean, mse_nw_mean, mse_ss_mean)
)

# Print the summary statistics
print(summary_stats)
```

```
#PART 2 ____________________________________________________________________

set.seed(79)

## True function (f(x)) definition (eq. (2) as given)
f_true2 <- function(x) (1 - x2^2) * exp(-0.5 * x2^2)

## Number of simulations (Monte Carlo runs)
m <- 1000
n <- 101
x2 <- round(2*pi*sort(c(0.5, -1 + rbeta(50,2,2), rbeta(50,2,2))), 8)

length(x2)
summary(x2)
plot_histogram(x2)
plot_qq(x2)


## Initialize the matrix of fitted values for three methods
fvlp2 <- fvnw2 <- fvss2 <- matrix(0, nrow=n, ncol=m)

## Run the simulations
for (j in 1:m) {
  ## Simulate y-values based on true function and noise
  y <- (1-x2^2) * exp(-0.5 * x2^2) + rnorm(length(x2), sd=0.2);
  fvlp2[,j] <- predict(loess(y ~ x2, span = 0.3365), newdata = x2);
```

```
 fvnw2[,j] <- ksmooth(x2, y, kernel="normal", bandwidth= 0.2, x.points=x2)$y;
 fvss2[,j] <-  predict(smooth.spline(y ~ x2, spar= 0.7163), x=x2)$y
}

## Initialize vectors to store the empirical bias, variance, and MSE for each method
bias_lp2 <- variance_lp2 <- mse_lp2 <- numeric(n)
bias_nw2 <- variance_nw2 <- mse_nw2 <- numeric(n)
bias_ss2 <- variance_ss2 <- mse_ss2 <- numeric(n)

## Compute bias, variance, and MSE for each method at each x_i
for (i in 1:n) {
 ## Collect the fitted values for the current x_i across all simulations
 y_lp2 <- fvlp2[i, ]
 y_nw2 <- fvnw2[i, ]
 y_ss2 <- fvss2[i, ]

 ## True value at x_i
 true_value2 <- f_true2(x[i])

 ## Compute the mean of the fitted values (fm(xi))
 mean_lp2 <- mean(y_lp2)
 mean_nw2 <- mean(y_nw2)
 mean_ss2 <- mean(y_ss2)

 ## Compute Bias (average of estimates - true value)
 bias_lp2[i] <- mean_lp2 - true_value2
 bias_nw2[i] <- mean_nw2 - true_value2
```

```
 bias_ss2[i] <- mean_ss2 - true_value2

 ## Compute Variance (variance of estimates)
 variance_lp2[i] <- mean((y_lp2 - mean_lp2)^2)
 variance_nw2[i] <- mean((y_nw2 - mean_nw2)^2)
 variance_ss2[i] <- mean((y_ss2 - mean_ss2)^2)

 ## Compute MSE (mean squared error between estimates and true value)
 mse_lp2[i] <- mean((y_lp2 - true_value2)^2)
 mse_nw2[i] <- mean((y_nw2 - true_value2)^2)
 mse_ss2[i] <- mean((y_ss2 - true_value2)^2)
}

## Below is the sample R code to plot the mean of three estimators in a single plot
meanlp2 = apply(fvlp2,1,mean);
meannw2 = apply(fvnw2,1,mean);
meanss2 = apply(fvss2,1,mean);
dmin2 = min( meanlp2,  meannw2,  meanss2);
dmax2 = max( meanlp2,  meannw2,  meanss2);
matplot(x2, meanlp2, "l", ylim=c(dmin2, dmax2), ylab="Response")
matlines(x2, meannw2, col="red")
matlines(x2, meanss2, col="blue")
legend("topright", legend=c("LOESS", "Kernel", "Spline"), col=c("black", "red", "blue"), lty=1)

## You might add the raw observations to compare with the fitted curves
points(x,y)
```

```
## Plot the empirical bias, variance, and MSE


par(mar=c(5, 5, 4, 6))


## Plot Bias

plot(x2, bias_lp2, type="l", col="black", ylim=c(min(bias_lp2, bias_nw2, bias_ss2), max(bias_lp2, bias_nw2, bias_ss2)),

    ylab="Bias", xlab="x2", main="Bias for Different Methods")

lines(x2, bias_nw2, col="red")

lines(x2, bias_ss2, col="blue")

legend("topright", legend=c("LOESS", "Kernel", "Spline"), col=c("black", "red", "blue"), lty=1)


## Plot Variance

plot(x2, variance_lp2, type="l", col="black", ylim=c(min(variance_lp2, variance_nw2, variance_ss2), max(variance_lp2, variance_nw2, variance_ss2)),

    ylab="Variance", xlab="x2", main="Variance for Different Methods")

lines(x2, variance_nw2, col="red")

lines(x2, variance_ss2, col="blue")

legend("topright", legend=c("LOESS", "Kernel", "Spline"), col=c("black", "red", "blue"), lty=1)


## Plot MSE

plot(x2, mse_lp2, type="l", col="black", ylim=c(min(mse_lp2, mse_nw2, mse_ss2), max(mse_lp2, mse_nw2, mse_ss2)),

    ylab="MSE", xlab="x2", main="MSE for Different Methods")

lines(x2, mse_nw2, col="red")

lines(x2, mse_ss2, col="blue")

legend("topright", legend=c("LOESS", "Kernel", "Spline"), col=c("black", "red", "blue"), lty=1)
```

```
# Now compute summary statistics for each method
bias_lp_mean2 <- mean(bias_lp2)
var_lp_mean2 <- mean(variance_lp2)
mse_lp_mean2 <- mean(mse_lp2)

bias_nw_mean2 <- mean(bias_nw2)
var_nw_mean2 <- mean(variance_nw2)
mse_nw_mean2 <- mean(mse_nw2)

bias_ss_mean2 <- mean(bias_ss2)
var_ss_mean2 <- mean(variance_ss2)
mse_ss_mean2 <- mean(mse_ss2)

# Create a data frame to display the results in a readable format
summary_stats2 <- data.frame(
  Method = c("LOESS", "Kernel Smoothing", "Spline Smoothing"),
  Bias = c(bias_lp_mean2, bias_nw_mean2, bias_ss_mean2),
  Variance = c(var_lp_mean2, var_nw_mean2, var_ss_mean2),
  MSE = c(mse_lp_mean2, mse_nw_mean2, mse_ss_mean2)
)

# Print the summary statistics
print(summary_stats2)
```