Homework-4

Introduction

Figure 1A in the Appendix shows a plot of data points, equidistant or non-equidistant, to be used in analyzing the statistical properties and computational challenges of local smoothing such as (1) LOESS, (2) Nadaraya-Watson (NW) kernel smoothing, and (3) spline smoothing. Figure 1B (Appendix) shows the same data points but now transformed by the so-called "Mexican Hat" function. Note that Figure 1A (and 1B) shows a single set of data points only.

A **spline** smoother uses a range of the available `x` values to determine its smoothness while **kernel** smoothing uses a weighted average of the corresponding `x/y` point. **LOESS** (and LOWESS) is a non-parametric, locally-weighted regression using the nearest neighbor approach.

Methods

A synthetic set of equidistant or non-equidistant data points was generated using a domain of $[-2\pi, 2\pi]$ and transformed with the so-called Mexican hat function. Subsequently, a random error was added to each data point where the errors were iid $N(0, 0.2^2)$.

The smoothing methods and associated parameters for equidistant data points were LOESS with span=0.75, Nadaraya-Watson (NW) kernel smoothing with a Gaussian kernel and bandwidth=0.2, and spline smoothing with the default tuning parameter.

The smoothing methods and associated parameters for non-equidistant data points were LOESS with span=0.3365, NW kernel smoothing with a Gaussian kernel and bandwidth=0.2, and spline smoothing with spar=0.7163.

For LOESS, the `span` (0-1) parameter controls the degree of smoothing with higher values resulting in a greater smoothing. Similarly, the `bandwidth` parameter used with `ksmooth` controls the degree of smoothing with higher values resulting in greater smoothing. Spline smoothing is accomplished with `smooth.spline` which uses the `spar` parameter to control the degree of smoothing.

Results

Equidistant data points Figure 2 (Appendix) shows the results of applying the different smoothing functions with equidistant data points. Clearly, LOESS applies a significant amount of smoothing while the kernel and spline smoothing methods apply very little, if any, smoothing at all.

Figures 3A, 3B, and 3C in the Appendix show the calculated bias, variance, and mean squared error (MSE) calculated at each `x` in the transformed dataset for each of the three smoothing functions following a Monte Carlo simulation (n=1000). The mean absolute bias with LOESS is highest at x = 0 (the "tip" of the Mexican hat) and decreases to zero as the **absolute** value of x increases to approximately one after which the absolute bias increases again until the absolute value of x is approximately 2. A further increase in the absolute value of x shows a decreasing bias towards zero until the absolute value of x is approximately three. The absolute bias then increases again slightly until the absolute value of x is approximately 4 after which it increases again albeit less so as observed in the [0,1] range. Note that the bias pattern is symmetrical due to the underlying smoothing applied. In contrast, the bias for both kernel and spline smoothing is (very) close to zero. Note that, with LOESS, the bias is negative middle in the middle of the curve (the "tip" of the Mexican hat)

and positive when x is approximately -1.8 or 1.8 (the "dips" in the Mexican hat) with intermediate values in between. Overall, depending on the value of x, the bias ranges from approximately -0.75 to 0.50. With respect to the variance, a similar result is obtained except that, obviously, the variance is always positive. The variance for LOESS is again highest in the middle of the curve with two flanking peaks of somewhat lower variance (Figure 3B). The variance for the other two methods, kernel and spline smoothing) is nearly zero over the range of x values studied. The results for the MSE are shown in Figure 3C (Appendix) and mirror the results for the variance except that the squared value of the bias has been added to the variance.

Non-equidistant data points Figure 4 (Appendix) shows the results of applying the different smoothing functions with **non**-equidistant data points. Here, the smoothing parameters have been chosen to allow for a more meaningful comparison between the different methods. Note that the data points are space further apart in the middle of the Mexican hat function and, as such, it expected that smoothing will be most affected by the choice of the appropriate smoothing parameter.

Figures 5A, 5B, and 5C in the Appendix show the calculated bias, variance, and MSE calculated at each `x` in the transformed dataset for each of the three smoothing functions following a Monte Carlo simulation (n=1000). The absolute bias was with LOES was again highest in the middle of the function but only approximately one-third of that observed with equidistant data points. Please note that this difference in bias is not (entirely) related to the fact that the data points in Figures 5 are non-equidistant. The choice of smoothing parameters has a much greater influence. Additionally, it should be noted that the variance with non-equidistant data points is lower by an order of magnitude as compared to the variance seen with equidistant data points.

The importance of the smoothing parameter is exemplified in Figures 6 and 7 (Appendix). An increase in the value of `span` for LOESS results in a higher SSE because higher `span` values result in more smoothing. It is noteworthy that the SSE for non-equidistant data points is generally lower as compared to equidistant data points (Figure 6). The results in Figure 7 show that a low span value (0.2) results in overfitting (similar to the use case with equidistant data points) while a relatively high span value (0.9) results in significant underfitting.

Findings (Discussion)

A fair comparison between different smoothing methods is only possible when the smoothing parameters are set in such a way that all methods apply smoothing to a similar degree. In other words, when the local neighborhood considered to derive the values between neighbors is quantitatively similar. The `span` parameter for LOESS is set rather high (with equidistant data points) and the regression is "over-smoothed" as compared to kernel and spline smoothing the parameters of which were set at a more reasonable value necessary to adequately estimate the underlying function. Both quantitative and qualitative differences were noted for the bias and variance between the two data sets, i.e. between the function values obtained with equidistant or non-equidistant data points. However, as noted above, a meaningful comparison between equidistant and non-equidistant data points is not quite possible since the smoothing parameters are substantially different and, as such, yield qualitatively different results. As with any other method used to estimate the underlying function, smoothing is prone to underfitting and overfitting (see Figure 7) and, as such, the smoothing parameters should be carefully chosen. The final choice of smoothing parameter depends on the bias-variance trade-off which is likely to be different for different types of data. Using the Mexican hat function it was observed that the absolute bias was relatively high when **high** values were chosen for the smoothing parameters. This is not unexpected as more smoothing will inevitable increase the bias (see Figure 6). On the other hand, a very low value for the smoothing parameter, resulting in a curve closely resembling the original function will yield a relatively low bias but higher variance. Thus, the "optimal" choice of smoothing parameters depends on an acceptable level of bias where a lower level of bias (better performance of the training data) is associated with higher bias (worse performance on unseen test data) and vice versa.

In general, kernel density smoothing is similar to a moving average hat its average is weighted and has a

fixed bin width and is relatively slow. LOESS extends the idea of fitting a line over variable bin-widths but it is a weighted regression line. It is computationally expensive (O^2)although this is not really concern using a computer. Spline smoothing cuts the domain of Y into partitions over which the algorithm computes a spline which are subsequently joined at intersections called knots. A short description of computational considerations can be found in "The Elements of Statistical Learning" (p. 216).

Appendix

Define required functions.

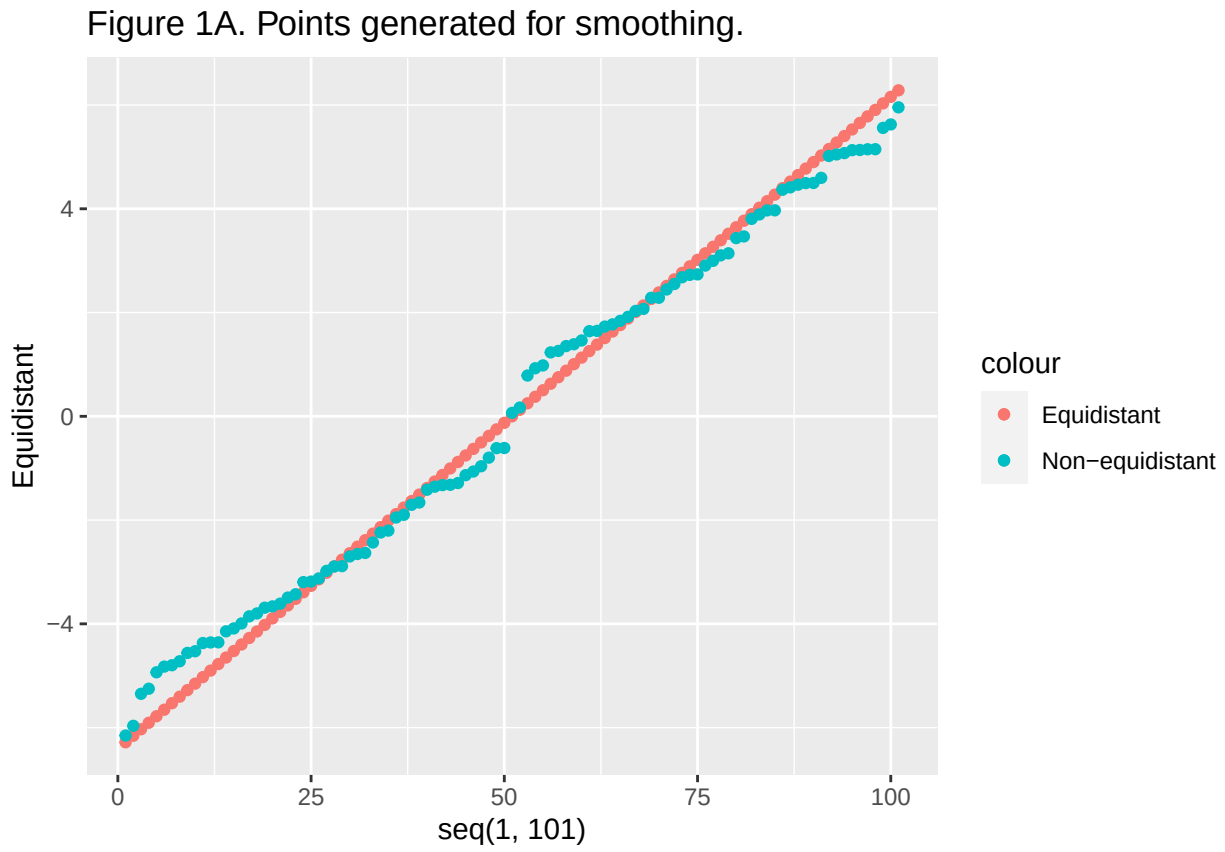
```
mex_hat <- function(x){
  res <- (1-x**2)*exp(-0.5*x**2)
  return( res)
}
```

```
gen_equidist_points <- function(n){
  res <- 2*pi*seq(-1, 1, length=n)
  return(res)
}

gen_nonequidist_points <- function(){
  set.seed(79)
  x <- 2*pi*sort(c(0.5, -1+rbeta(50,2,2), rbeta(50,2,2)))
  return(x)
}

equi_dist_x <- gen_equidist_points(101)
non_equi_dist_x <- gen_nonequidist_points()
```

```
res1 <- as.data.frame(equi_dist_x)
res2 <- as.data.frame(non_equi_dist_x)
X <- cbind(res1, res2)
colnames(X) <- c("Equidistant", "Non_equidistant")
```

The plot below shows the values of the data points generated

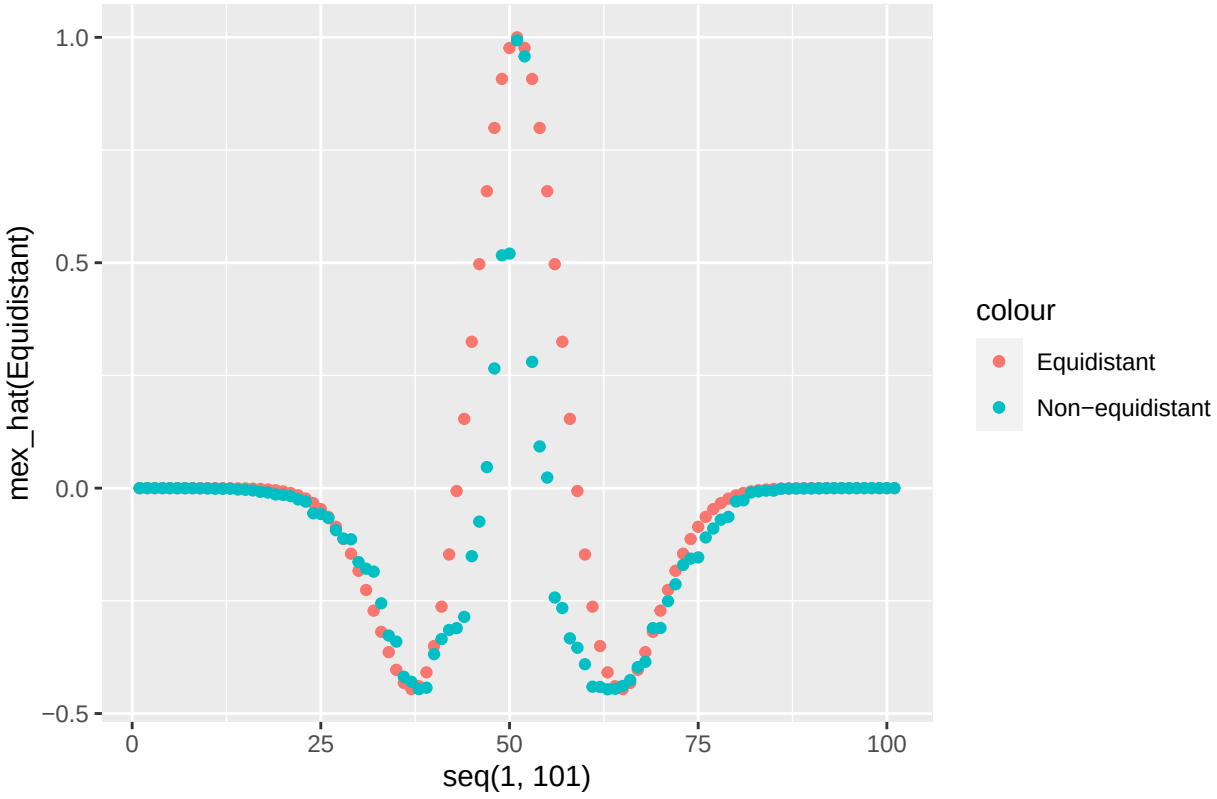
```
ggplot(data=X, aes(x=seq(1,101), y=Equidistant, color="Equidistant"))+
  geom_point()+
  geom_point(aes(x=seq(1,101), y=Non_equidistant, color="Non-equidistant"))+
  labs(title="Figure 1A. Points generated for smoothing.",
       xlab="Index",
       ylab="Value")+
  theme(legend.position="right")
```



Plot of data points transformed with the Mexican Hat function.

```
ggplot(data=X, aes(x=seq(1,101), y=mex_hat(Equidistant), color="Equidistant"))+
  geom_point()+
  geom_point(aes(x=seq(1,101), y=mex_hat(Non_equidistant), color="Non-equidistant"))+
  labs(title="Figure 1B. Points generated for smoothing after 'Mexican Hat' function.",
       xlab="Index",
       ylab="Value")+
  theme(legend.position="right")
```

Figure 1B. Points generated for smoothing after 'Mexican Hat' function.



Define a function that generates the fitted values for a dataset.

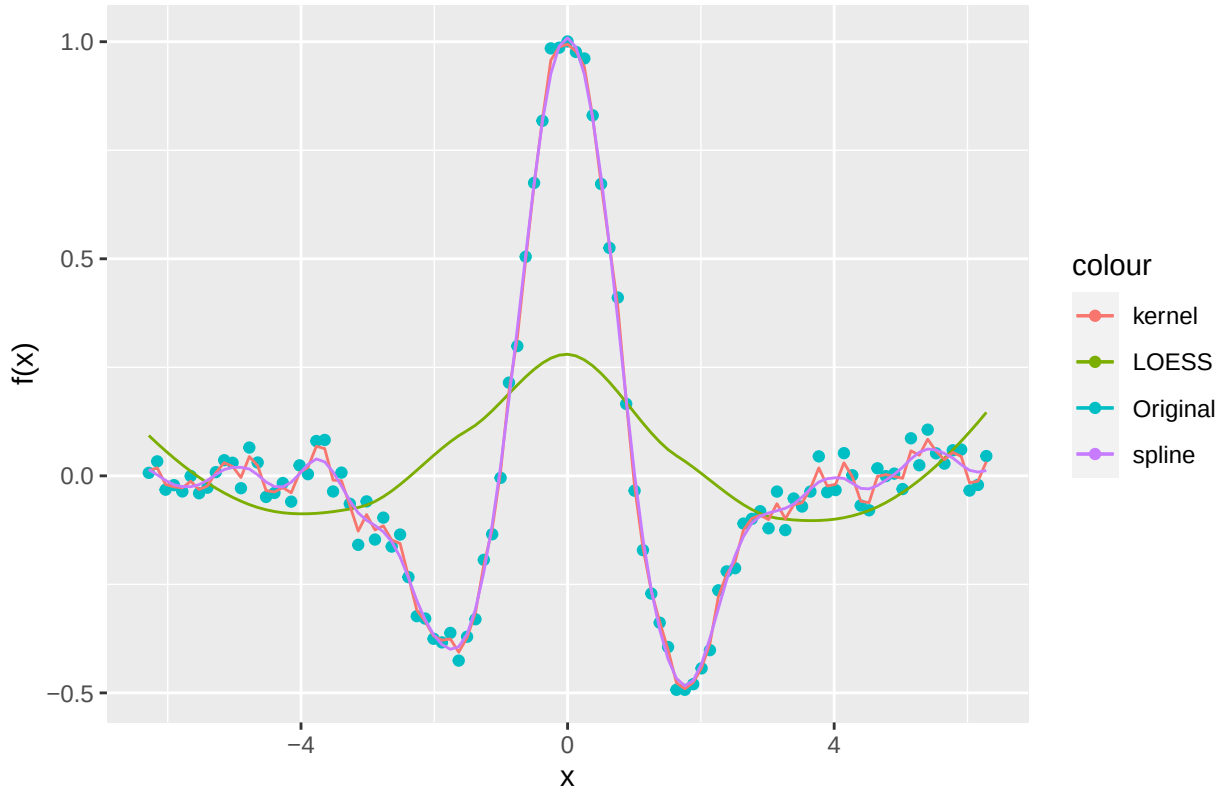
```
get_fitted_values <- function(FUN=mex_hat, x){
  ## Uses a function FUN to transform a set of equidistant points
  ## then adds an error term
  n = length(x)
  eps <- rnorm(n, 0, 0.2**2) # epsilon
  y <- FUN(x) + eps
  return(y)
}

y_equi <- get_fitted_values(mex_hat, equi_dist_x) # the 'true' values
y_non_equi <- get_fitted_values(mex_hat, non_equi_dist_x)
```

Plot predictions of different smoothing functions using **equidistant** data points.

```
loess_pred <- predict(loess(y_equi ~ equi_dist_x, span=0.75), newdata=equi_dist_x)
kernel_pred <- ksmooth(equi_dist_x, y_equi, kernel="normal", bandwidth=0.2, x.points=equi_dist_x)$y
spline_pred <- predict(smooth.spline(y_equi ~ equi_dist_x), spar=NULL, x=equi_dist_x)$y

ggplot(data=NULL, aes(x=equi_dist_x, y=y_equi, color="Original"))+
  geom_point()+
  geom_line(aes(x=equi_dist_x, y=loess_pred, color="LOESS"))+
  geom_line(aes(x=equi_dist_x, y=kernel_pred, color="kernel"))+
  geom_line(aes(x=equi_dist_x, y=spline_pred, color="spline"))+
  labs(title="Figure 2. Smoothing on a single fitted dataset.",
       x="x",
```

```
      y="f(x)")
```

Figure 2. Smoothing on a single fitted dataset.



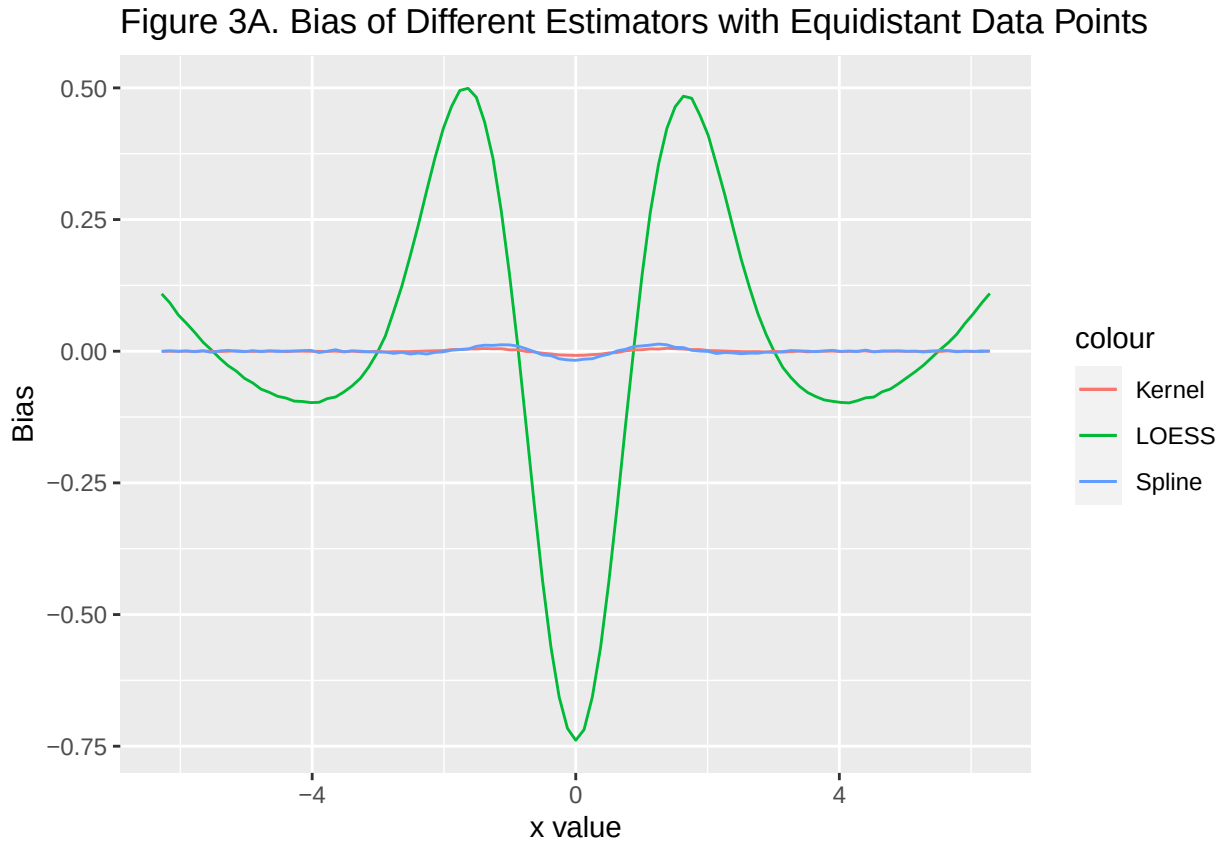
Equidistant data points Define a function to perform Monte Carlo simulation.

```
run_mc <- function(input, n_sims, sm_type="LOESS", span=0.75, bandwidth=0.2, spar=NULL){

  # Function returns a matrix where:
  # each row represents a data point

  # Initialize matrix of fitted values
  bias_matrix <- matrix(0, nrow=length(input), ncol=n_sims)
  var_matrix <- matrix(0, nrow=length(input), ncol=n_sims)

  for(i in 1:n_sims){
    # Obtain fitted values
    y <- get_fitted_values(FUN=mex_hat, input)
    # Get predictions
    if(sm_type == "LOESS"){
      predictions <- predict(loess(y ~ input, span=span), newdata=input)
      bias_matrix[, i] <- predictions - y
      var_matrix[, i] <- (predictions - y)**2
    } else if(sm_type=="kernel"){
      predictions <- ksmooth(input, y, kernel="normal", bandwidth=bandwidth, x.points=input)$y
      bias_matrix[, i] <- predictions - y
      var_matrix[, i] <- (predictions - y)**2
```

```
    } else if(sm_type=="spline"){
      predictions <- predict(smooth.spline(y ~ input), x=input, spar=spar)$y
      bias_matrix[, i] <- predictions - y
      var_matrix[, i] <- (predictions - y)**2
    }
  }

  bias <- apply(bias_matrix, 1, mean)
  variance <- apply(var_matrix, 1, mean)

  return(list(bias=bias, variance=variance))
}

n_sims <- 1000
res_loess <- run_mc(equi_dist_x, n_sims, sm_type="LOESS")
res_kernel <- run_mc(equi_dist_x, n_sims, sm_type="kernel")
res_spline <- run_mc(equi_dist_x, n_sims, sm_type="spline")
```

Calculate and plot the empirical bias of the three estimators (using **equidistant** data points)

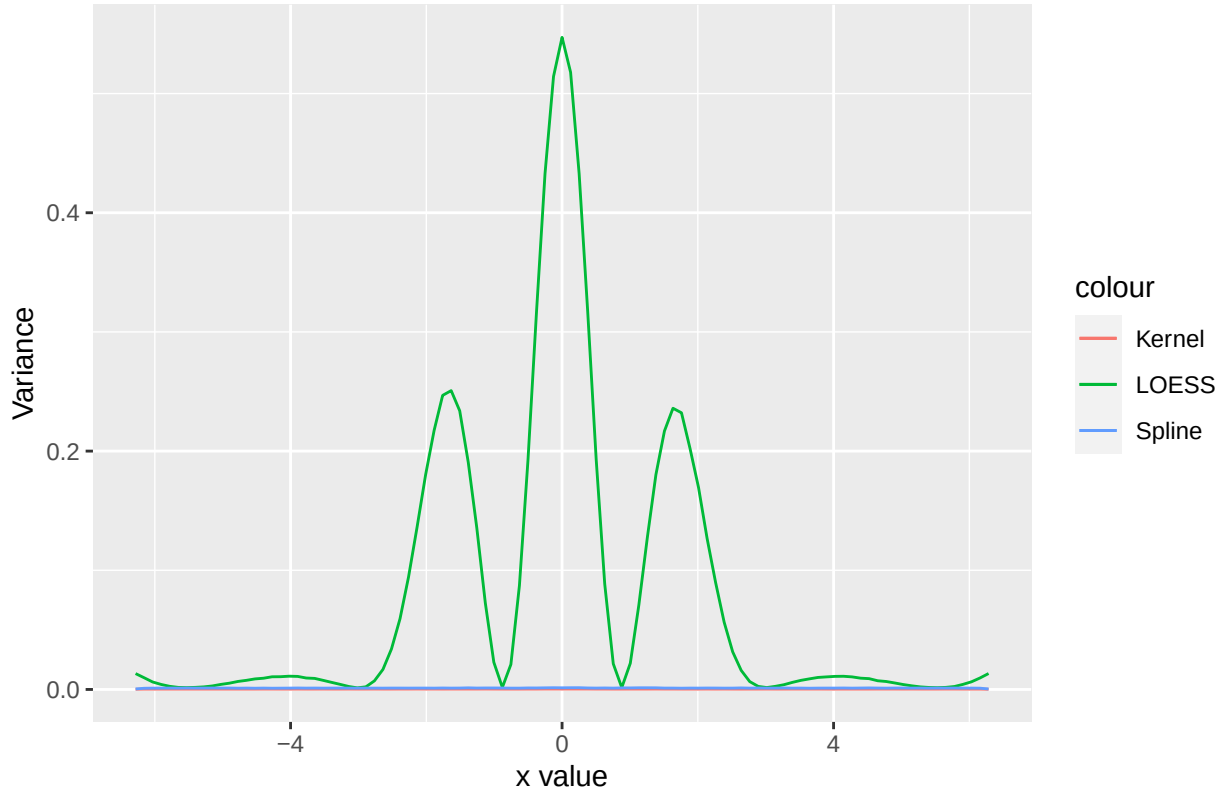
```
b_loess <- res_loess$bias
b_kernel <- res_kernel$bias
b_spline <- res_spline$bias

ggplot(data=NULL, aes(x=equi_dist_x, y=b_loess, color="LOESS"))+
  geom_line()+
  geom_line(aes(x=equi_dist_x, y=b_kernel, color="Kernel"))+
  geom_line(aes(x=equi_dist_x, y=b_spline, color="Spline"))+
  labs(title="Figure 3A. Bias of Different Estimators with Equidistant Data Points",
       x="x value",
       y="Bias")
```



Calculate and plot the empirical variance of the three estimators.

```
v_loess <- res_loess$variance
v_kernel <- res_kernel$variance
v_spline <- res_spline$variance

ggplot(data=NULL, aes(x=equi_dist_x, y=v_loess, color="LOESS"))+
  geom_line()+
  geom_line(aes(x=equi_dist_x, y=v_kernel, color="Kernel"))+
  geom_line(aes(x=equi_dist_x, y=v_spline, color="Spline"))+
  labs(title="Figure 3B. Variance of Different Estimators with Equidistant Data Points",
       x="x value",
       y="Variance")
```

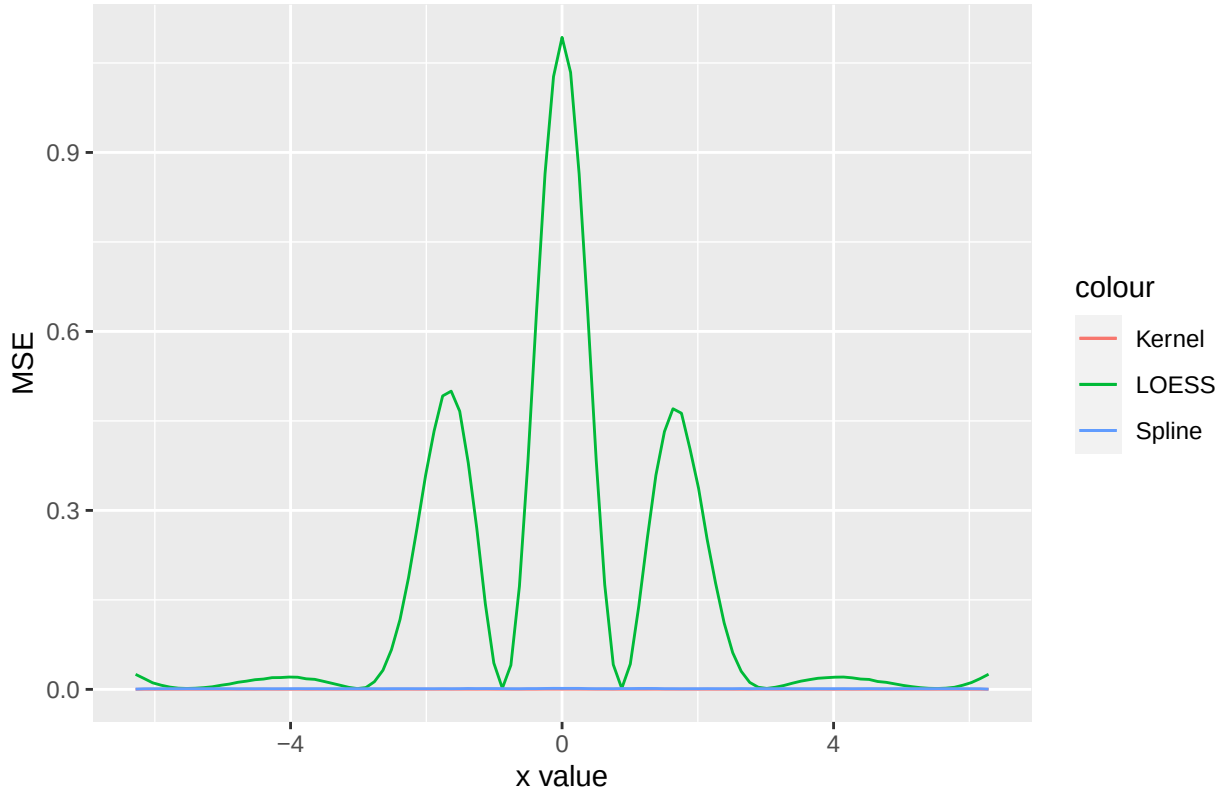
Figure 3B. Variance of Different Estimators with Equidistant Data Points



Calculate and plot empirical MSE of the three estimators.

```
mse_loess <- b_loess**2 + v_loess
mse_kernel <- b_kernel**2 + v_kernel
mse_spline <- b_spline**2 + v_spline

ggplot(data=NULL, aes(x=equi_dist_x, y=mse_loess, color="LOESS"))+
  geom_line()+
  geom_line(aes(x=equi_dist_x, y=mse_kernel, color="Kernel"))+
  geom_line(aes(x=equi_dist_x, y=mse_spline, color="Spline"))+
  labs(title="Figure 3C. MSE of Different Estimators with Equidistant Data Points",
       x="x value",
       y="MSE")
```

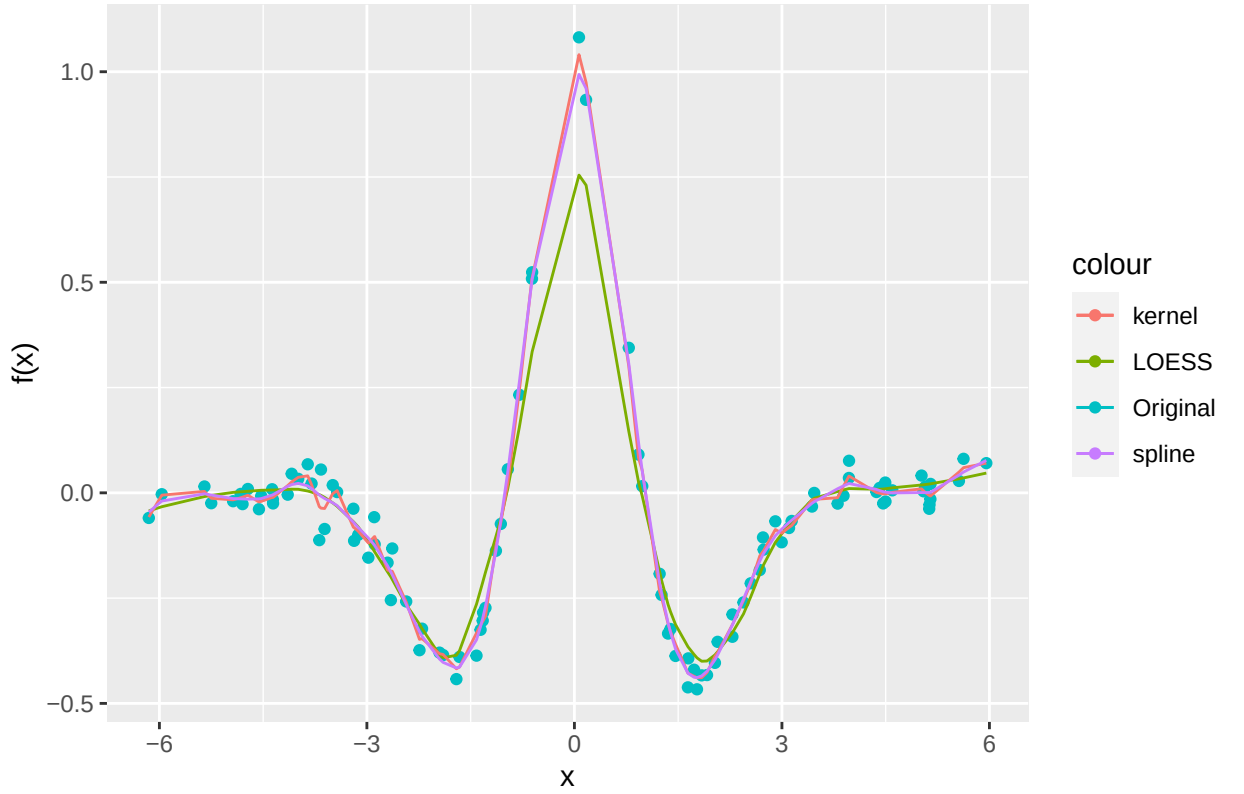
Figure 3C. MSE of Different Estimators with Equidistant Data Points



Non-equidistant data points Plot predictions of different smoothing functions using **non-equidistant** data points.

```
loess_pred2 <- predict(loess(y_non_equi ~ non_equi_dist_x, span=0.3365), newdata=non_equi_dist_x)
kernel_pred2 <- ksmooth(non_equi_dist_x, y_non_equi, kernel="normal", bandwidth=0.2, x.points=non_equi_
spline_pred2 <- predict(smooth.spline(y_non_equi ~ non_equi_dist_x), spar=0.7163, x=non_equi_dist_x)$y

ggplot(data=NULL, aes(x=non_equi_dist_x, y=y_non_equi, color="Original"))+
  geom_point()+
  geom_line(aes(x=non_equi_dist_x, y=loess_pred2, color="LOESS"))+
  geom_line(aes(x=non_equi_dist_x, y=kernel_pred2, color="kernel"))+
  geom_line(aes(x=non_equi_dist_x, y=spline_pred2, color="spline"))+
  labs(title="Figure 4. Smoothing on a single fitted dataset with non-equidistant data points.",
       x="x",
       y="f(x)")
```

Figure 4. Smoothing on a single fitted dataset with non−equidistant data pc

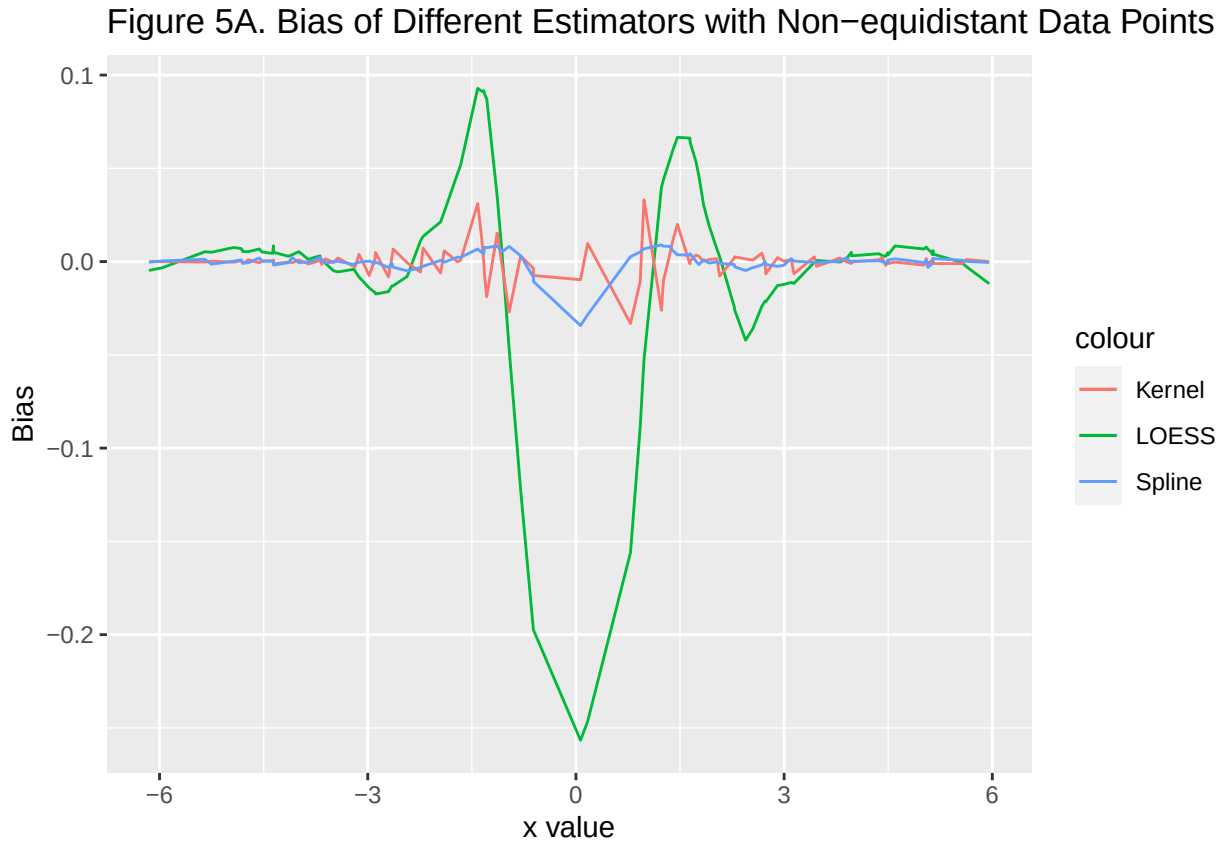


Run MC simulations

```
n_sims <- 1000
res2_loess <- run_mc(non_equi_dist_x, n_sims, sm_type="LOESS", span=0.3365, bandwidth=0.2, spar=0.7163)
res2_kernel <- run_mc(non_equi_dist_x, n_sims, sm_type="kernel", span=0.3365, bandwidth=0.2, spar=0.716
res2_spline <- run_mc(non_equi_dist_x, n_sims, sm_type="spline", span=0.3365, bandwidth=0.2, spar=0.716
```

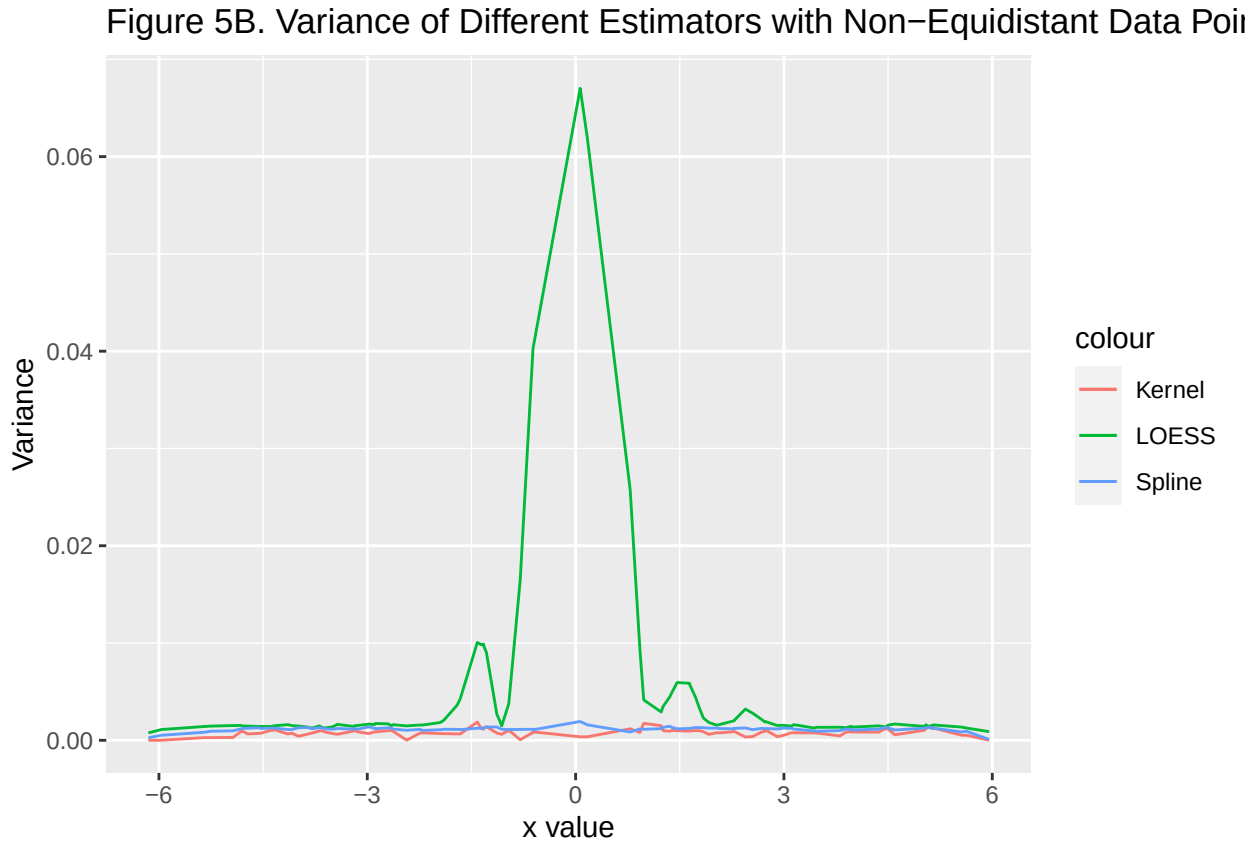
Calculate and plot the empirical bias of the three estimators

```
b2_loess <- res2_loess$bias
b2_kernel <- res2_kernel$bias
b2_spline <- res2_spline$bias

ggplot(data=NULL, aes(x=non_equi_dist_x, y=b2_loess, color="LOESS"))+
  geom_line()+
  geom_line(aes(x=non_equi_dist_x, y=b2_kernel, color="Kernel"))+
  geom_line(aes(x=non_equi_dist_x, y=b2_spline, color="Spline"))+
  labs(title="Figure 5A. Bias of Different Estimators with Non-equidistant Data Points",
       x="x value",
       y="Bias")
```



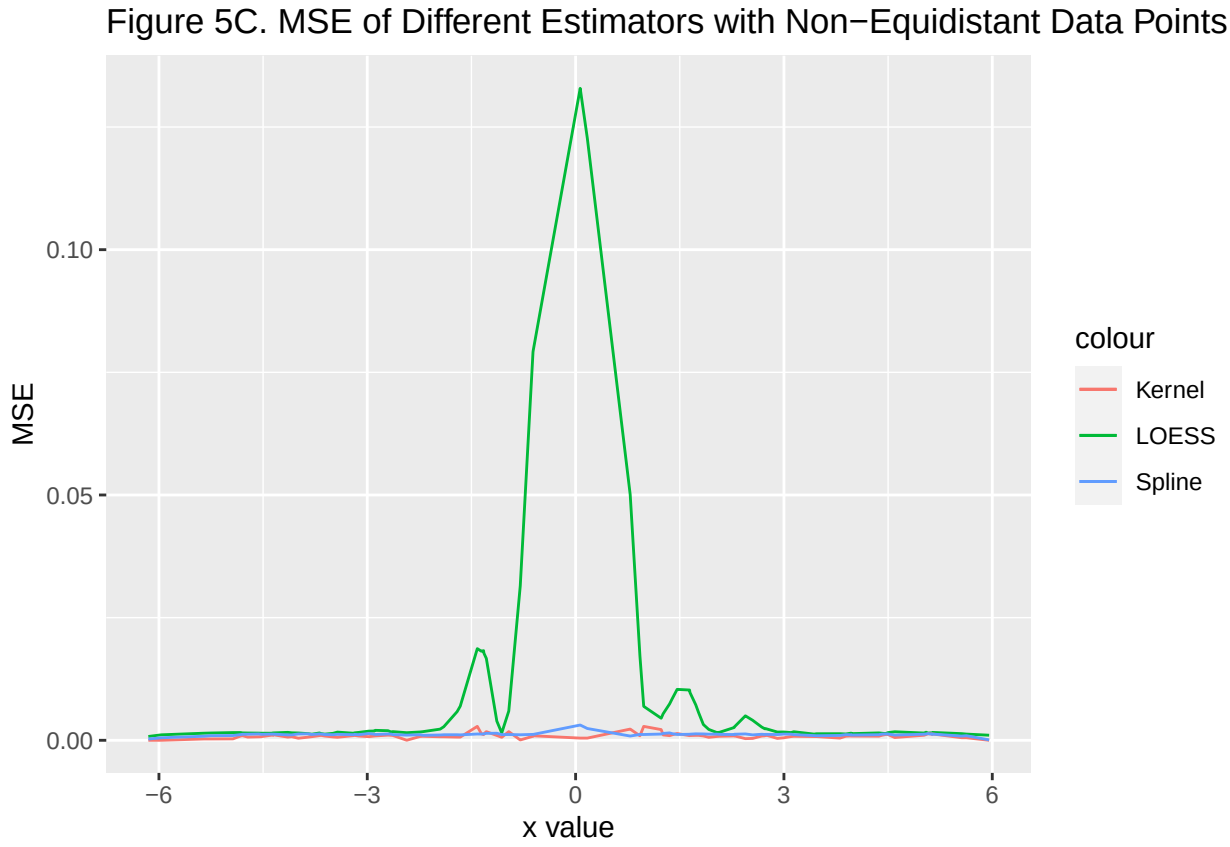
Calculate and plot empirical variance of the three estimators.

```
v2_loess <- res2_loess$variance
v2_kernel <- res2_kernel$variance
v2_spline <- res2_spline$variance

ggplot(data=NULL, aes(x=non_equi_dist_x, y=v2_loess, color="LOESS"))+
  geom_line()+
  geom_line(aes(x=non_equi_dist_x, y=v2_kernel, color="Kernel"))+
  geom_line(aes(x=non_equi_dist_x, y=v2_spline, color="Spline"))+
  labs(title="Figure 5B. Variance of Different Estimators with Non-Equidistant Data Points",
       x="x value",
       y="Variance")
```



Calculate and plot empirical MSE of the three estimators.

```
mse2_loess <- b2_loess**2 + v2_loess
mse2_kernel <- b2_kernel**2 + v2_kernel
mse2_spline <- b2_spline**2 + v2_spline

ggplot(data=NULL, aes(x=non_equi_dist_x, y=mse2_loess, color="LOESS"))+
  geom_line()+
  geom_line(aes(x=non_equi_dist_x, y=mse2_kernel, color="Kernel"))+
  geom_line(aes(x=non_equi_dist_x, y=mse2_spline, color="Spline"))+
  labs(title="Figure 5C. MSE of Different Estimators with Non-Equidistant Data Points",
       x="x value",
       y="MSE")
```

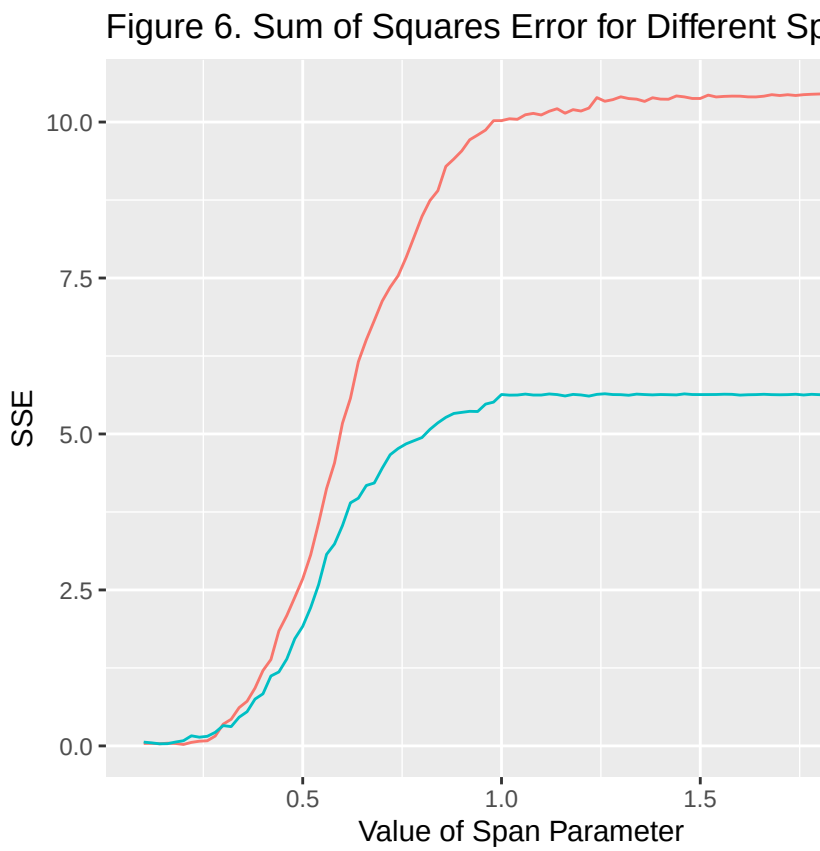


Define a function that returns the SSE for LOESS given a range of spans.

```
calc_sse <- function(input, span_seq){

  res <- NULL
  y_true <- mex_hat(input) # ground truth for function

  for(span in span_seq){
    y = get_fitted_values(mex_hat, input) # using mexican hat function

    loess_pred <- predict(loess(y ~ input, span=span), newdata=input)
    sse <- sum((loess_pred - y_true)**2)
    res <- c(res, sse)

  }
  return(res)
}

span_seq <- seq(0.1, 2, 0.02)

res_equi <- calc_sse(equi_dist_x, span_seq)
res_non_equi <- calc_sse(non_equi_dist_x, span_seq)
```

```
ggplot(data=NULL, aes(x=span_seq, y=res_equi, color="Equidistant"))+
    geom_line()+
```

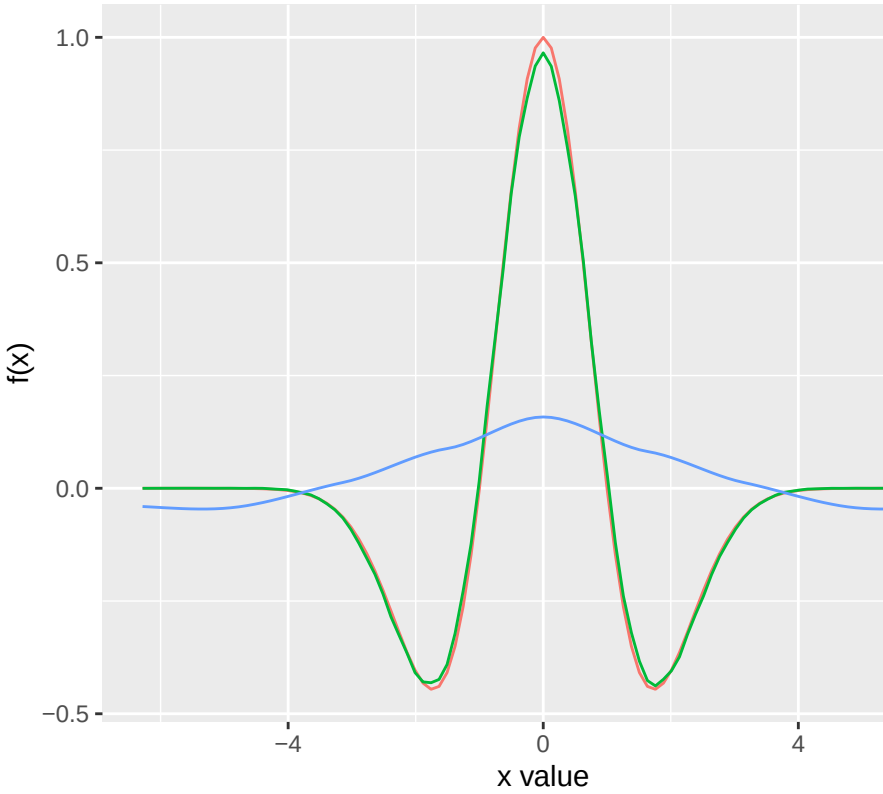
```
  geom_line(aes(x=span_seq, y=res_non_equi, color="Non-equidistant"))+
  labs(title="Figure 6. Sum of Squares Error for Different Span Values",
       x="Value of Span Parameter",
       y="SSE")
```



Plot SSE as a function of the span parameter.

```
x <- equi_dist_x
y <- mex_hat(x)

y_hat_1 <- predict(loess(y ~ x, span=0.2), newdata=x)
y_hat_9 <- predict(loess(y ~ x, span=0.9), newdata=x)

ggplot(data=NULL, aes(x=x, y=y, color="Original"))+
  geom_line()+
  geom_line(aes(x=x, y=y_hat_1, color="Span of 0.2"))+
  geom_line(aes(x=x, y=y_hat_9, color="Span of 0.9"))+
  labs(title="Figure 7. Overfitting and Underfitting with LOESS",
       x="x value",
       y="f(x)")
```

Figure 7. Overfitting and Underfitting with LOESS



Plot LOESS curves with different spans.

Literature

1. Deng, H. & Wickham, H. (2011) Density estimation in R. https://vita.had.co.nz/papers/density-estimation.pdf (retrieved March 6, 2021).
2. Luedicke, J. Friedman's Super Smoother. https://fmwww.bc.edu/repec/bocode/s/supsmooth_doc.pdf (rerieved March 6, 2021).
3. Hastie, T., Tibshirani, R. & Friedman, J. (2017) The Elements of Statistical Learning, $2^n d\}$ Ed., Springer