Solution to Series 3

1. **Note:** The electronic version of this document uses colors and may be easier to read.

   We first define a function which simulates data, computes the span (degree of smoothing) and spar (equivalent number of degrees of freedom) parameters, and calculates estimated means and standard errors for each simulation run. The argument x of the function are the design points. The function returns a list with the estimated means and estimated standard errors for the Nadaraya-Watson, the Local Polynomial, and the Smoothing Splines nonparametric regression estimate.

```
> simul <- function(x,m,nrep=1000){
    set.seed(79)

    n <- length(x)
    ## Prepare hat matrices
    Snw <- Slp <- Sss <- matrix(0, nrow = n, ncol = n)

    ## Calculate the hat matrix for Nadaraya-Watson, it only depends on x
    ## The j-th column is given by Se_j
    In <- diag(rep(1, n))
    for(j in 1:n){
       Snw[,j] <- ksmooth(x, In[,j], kernel = "normal", bandwidth = 0.2,
                          x.points = x)$y
    }

    ## Give out the degrees of freedom for Nadaraya-Watson estimator
    cat("Degrees of freedom for Nadaraya-Watson:",format(sum(diag(Snw))),"\n")

    ## Getting the span parameter for loess and
    ## spar parameter for smooth.spline such that the
    ## degrees of freedom are (approximately) the same
    ## with the ones for Nadaraya-Watson estimator

    dflp <- function(span, val){
      for(j in 1:n)
         Slp[,j] <- loess(In[,j] ~ x, span = span)$fitted
      return(sum(diag(Slp)) - val)
    }

    ## What span value leads to the desired df-value?
    span <- uniroot(dflp, c(0.2, 0.5), val = sum(diag(Snw)))$root
    ## Give out the span value for Local Polynomial regression estimator
    ## with same degrees of freedom
    cat("Span value for Local Polynomial:",format(span),"\n")

    ## Compute smoothing matrix using loess, respectively smooth.spline
    for(j in 1:n){
      Slp[,j] <- predict(loess(In[,j] ~ x, span = span), newdata = x)
      Sss[,j] <- predict(smooth.spline(x, In[,j], df = sum(diag(Snw))), x = x)$y
    }

    ## Get the spar value
    spar <- smooth.spline(x, In[,1], df = sum(diag(Snw)))$spar
    ## Give out the span value for Smoothing Splines regression
    ## estimator with same degrees of freedom
```

```
  cat("Spar value for Smoothing Splines:",format(spar),"\n")

  ## Save the results of each kernel estimator in a matrix
  ## rows are x-positions, columns are simulation runs
  estnw <- estlp <- estss <- matrix(0, nrow = n, ncol = nrep)
  senw <- selp <- sess <- matrix(0, nrow = n, ncol = nrep)

  for(i in 1:nrep){
    ## Simulate y-values
    y <- m(x) + rnorm(length(x))

    ## Get estimates for the mean function
    estnw[,i] <- ksmooth(x, y, kernel = "normal", bandwidth = 0.2, x.points = x)$y
    estlp[,i] <- predict(loess(y ~ x, span = span), newdata = x)
    estss[,i] <- predict(smooth.spline(x, y, spar = spar), x = x)$y

    ## Compute the estimated variance of the error
    sigmanw <- sum((y - estnw[,i])^2) / (length(y) - sum(diag(Snw)))
    sigmalp <- sum((y - estlp[,i])^2) / (length(y) - sum(diag(Slp)))
    sigmass <- sum((y - estss[,i])^2) / (length(y) - sum(diag(Sss)))

    ## Compute the standard error
    senw[,i] <- sqrt(sigmanw * diag(Snw %*% t(Snw)))
    selp[,i] <- sqrt(sigmalp * diag(Slp %*% t(Slp)))
    sess[,i] <- sqrt(sigmass * diag(Sss %*% t(Sss)))
  }
  ##Return the estimated means and the standarderrors in a list
  return(list(estnw=estnw,estlp=estlp,estss=estss,senw=senw,selp=selp,sess=sess))
}
```

a) Let us first consider the situation with equidistant $x_i, i = 1, \ldots, 101$.

```
> x <- seq(-1, 1, length = 101)
```

We simulate and fit nonparametric regression estimates with the above function "simul".

```
> m <- function(x) x + 4*cos(7*x)
> Equi <- simul(x,m=m)
```

```
Degrees of freedom for Nadaraya-Watson: 11.33399
Span value for Local Polynomial: 0.2971339
Spar value for Smoothing Splines: 0.623396
```

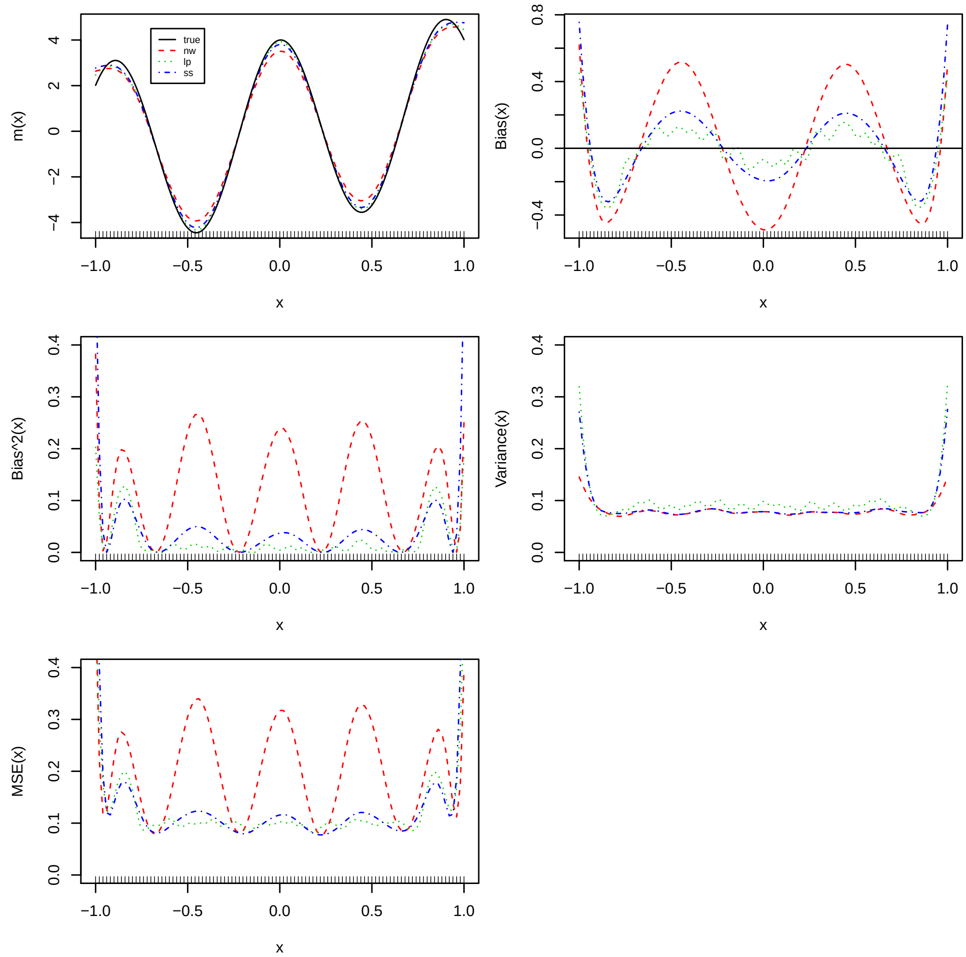
The results are stored in a list called Equi.

```
> str(Equi)
```

```
List of 6
 $ estnw: num [1:101, 1:1000] 2.65 2.57 2.48 2.37 2.26 ...
 $ estlp: num [1:101, 1:1000] 2.84 2.8 2.74 2.66 2.55 ...
 $ estss: num [1:101, 1:1000] 3.14 2.98 2.82 2.66 2.5 ...
 $ senw : num [1:101, 1:1000] 0.434 0.408 0.384 0.365 0.349 ...
 $ selp : num [1:101, 1:1000] 0.62 0.522 0.44 0.377 0.333 ...
 $ sess : num [1:101, 1:1000] 0.576 0.499 0.433 0.382 0.347 ...
```

We then calculate biases and variances with the following code.

```
> par(mfrow = c(3,2))
> means <- cbind(apply(Equi$estnw, 1, mean), apply(Equi$estlp, 1, mean),
                 apply(Equi$estss, 1, mean))
> matplot(x, means, lty = 2:4, col = 2:4, type = "l",
          xlab = "x", ylab = "m(x)")
> lines(seq(-1, 1, by = 0.01), m(seq(-1, 1, by = 0.01)))
> rug(x)
> legend(-0.7,4.5, lty = 1:4, col = 1:4, c("true", "nw", "lp", "ss"), cex = 0.6)
> bias <- means - m(x)
> matplot(x, bias, lty = 2:4, col = 2:4, type = "l",
          xlab = "x", ylab = "Bias(x)")
> rug(x)
> abline(h = 0)
> matplot(x, bias^2, lty = 2:4, col = 2:4, type = "l",
          xlab = "x", ylab = "Bias^2(x)", ylim = c(0, 0.4))
> rug(x)
> variances <- cbind(apply(Equi$estnw, 1, var), apply(Equi$estlp, 1, var),
                     apply(Equi$estss, 1, var))
> matplot(x, variances, lty = 2:4, col = 2:4, type = "l",
          xlab = "x", ylab = "Variance(x)", ylim = c(0, 0.4))
> rug(x)
> matplot(x, variances+bias^2, lty = 2:4, col = 2:4, type = "l",
          xlab = "x", ylab = "MSE(x)", ylim = c(0, 0.4))
> rug(x)
```



If we look at the mean of all simulations we see that the bias is high in **regions of curvature** of the true function (there is **"erosion"**). Infact, the bias is proportional to the curvature $m''(x)$. Comparing the different regression estimators, Local Polynomial (quadratic) and Smoothing Splines tend to have lower bias than Nadaraya-Watson. However, Local Polynomial has slightly higher variance than the other two but seems to have the lowest MSE.
The bias and variance are also high at the **boundaries**. For the bias, Local Polynomial performs best (it adapts best to the curvature of the true function at the boundaries) at the cost of highest variance. The Nadaraya-Watson estimator has a higher bias at the bondaries compared to the two other estimators. This is due to the fact that, in contrast to Local Polynomial and Smoothing Splines, the Nadaraya-Watson estimator makes only a locally constant fit.
Another possibility to visualize the variance is to draw all estimates (not only the pointwise means). This could be easily done in R with `matplot(x, estnw, type = "l", col = "grey")` (not shown here).

b) We first define a function which calculates how many times the pointwise confidence interval at $x = 0.5$ contains the true value and how many times the the confidence band for all points simultaneously contain all true values.

```
> coverage <- function(x,est,se){
    pos <- x == 0.5
    pw <- sum(abs(est[pos,] - m(x)[pos]) <= 1.96*se[pos,]) ## 622 or 793
    simult <- sum(apply(abs(est - m(x)) <= 1.96 * se, 2, all)) ## 9 or 1
    cat("Number of pointwise coverages:",pw,"\n")
    cat("Number of simultaneous coverages:",simult,"\n")
 }
```

We then calculate the coverages for the three nonparametric regression estimates. Nadaraya-Watson:

```
> coverage(x,Equi$estnw,Equi$senw)
```

```
Number of pointwise coverages: 622
Number of simultaneous coverages: 9
```

Local Polynomial:

```
> coverage(x,Equi$estlp,Equi$selp)
```

```
Number of pointwise coverages: 946
Number of simultaneous coverages: 157
```

Smoothing Spline:

```
> coverage(x,Equi$estss,Equi$sess)
```

```
Number of pointwise coverages: 895
Number of simultaneous coverages: 116
```

If we have a look at the (approximate) pointwise confidence interval at $x = 0.5$ we get a coverage rate of 622/1000 for Nadaraya-Watson, 946/1000 for Local Polynomial and 895/1000 for Smoothing Splines. The reason for this is that these are confidence intervals for $\mathbb{E}[\hat{m}(x)]$ and *not* for the *true* function value $m(x)$. You could subtract the (empirical) bias to correct the intervals (see manuscript). Of course the situation worsens when looking at the simultaneous confidence band. The combinations of the pointwise confidence interval is too optimistic as the coverage rates show: 9/1000, 157/1000, 116/1000 for Nadaraya-Watson, Local Polynomial and Smoothing Splines respectively. A simultaneous confidence band for the underlying function would be much wider.

c) In the case of non-equidistant x_i, we generate 101 "random" points according to 2 transformed beta(2,2) distributions.

```
> x <- sort(c(0.5, -1 + rbeta(50, 2, 2), rbeta(50, 2, 2)))
```

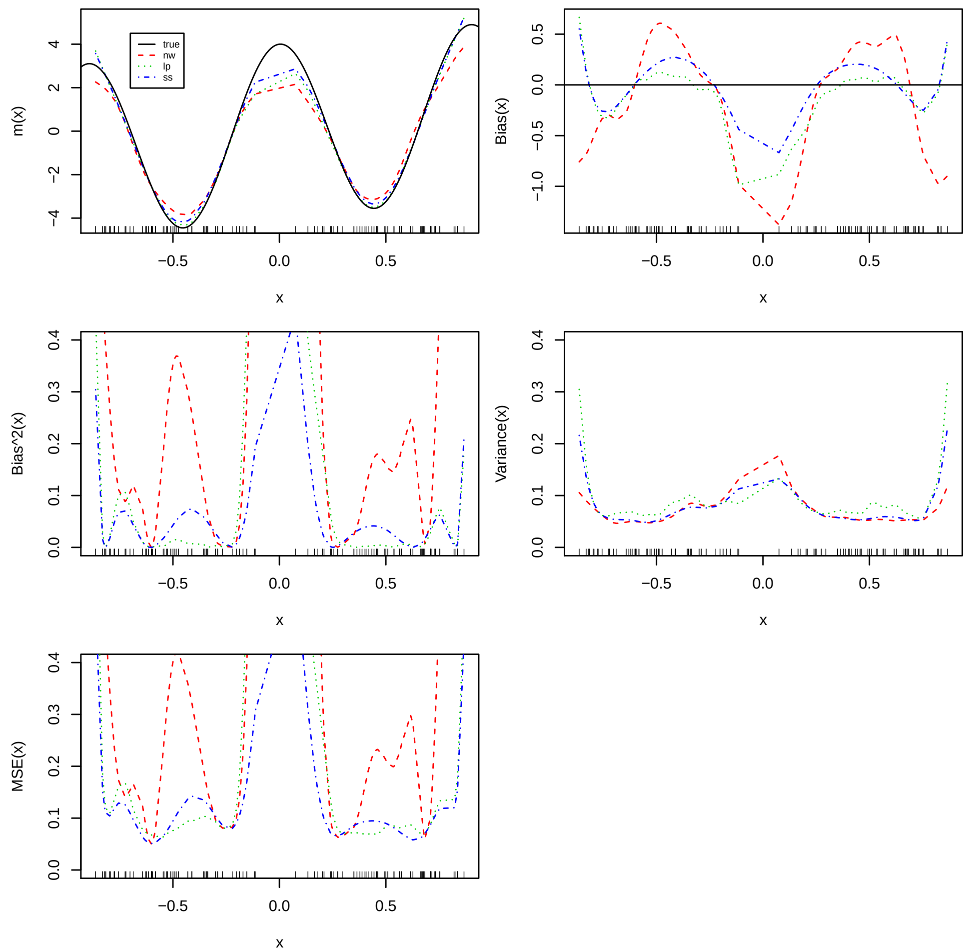
We then simulate and fit nonparametric regression estimates using the same function as before. The results are stored in a list called NonEqui.

```
> NonEqui <- simul(x,m=m)
```

```
Degrees of freedom for Nadaraya-Watson: 9.227774
Span value for Local Polynomial: 0.3761381
Spar value for Smoothing Splines: 0.7942459
```

Again, biases and variances are calculated and plotted.

```
> par(mfrow = c(3,2))
> means <- cbind(apply(NonEqui$estnw, 1, mean), apply(NonEqui$estlp, 1, mean),
                 apply(NonEqui$estss, 1, mean))
> matplot(x, means, lty = 2:4, col = 2:4, type = "l",
          xlab = "x", ylab = "m(x)")
> lines(seq(-1, 1, by = 0.01), m(seq(-1, 1, by = 0.01)))
> rug(x)
> legend(-0.7,4.5, lty = 1:4, col = 1:4, c("true", "nw", "lp", "ss"), cex = 0.6)
> bias <- means - m(x)
> matplot(x, bias, lty = 2:4, col = 2:4, type = "l",
          xlab = "x", ylab = "Bias(x)")
> rug(x)
> abline(h = 0)
> matplot(x, bias^2, lty = 2:4, col = 2:4, type = "l",
          xlab = "x", ylab = "Bias^2(x)", ylim = c(0, 0.4))
> rug(x)
> variances <- cbind(apply(NonEqui$estnw, 1, var), apply(NonEqui$estlp, 1, var),
                     apply(NonEqui$estss, 1, var))
> matplot(x, variances, lty = 2:4, col = 2:4, type = "l",
          xlab = "x", ylab = "Variance(x)", ylim = c(0, 0.4))
> rug(x)
> matplot(x, variances+bias^2, lty = 2:4, col = 2:4, type = "l",
          xlab = "x", ylab = "MSE(x)", ylim = c(0, 0.4))
> rug(x)
```



We see more or less the same effects as in a). There is a large bias in regions of curvature and at the boundaries.
The situation around 0 is special because we only have few points. This results in increased bias and variance. Nadaraya-Watson has highest bias and also high variance. Smoothing Splines is a bit worse than Local Polynomial in the bias but has lower variance. This is because you can think of Smoothing Spline as a kernel estimator with varying bandwidth. It automatically increases the bandwidth in regions where observations are sparse (see also the example in the manuscript).
Numbers of coverages, pointwise as well as simultaneuos, are again calculated with the function "coverage".
Nadaraya-Watson:

```
> coverage(x,NonEqui$estnw,NonEqui$senw)
```

```
Number of pointwise coverages: 666
Number of simultaneous coverages: 0
```

Local Polynomial:

```
> coverage(x,NonEqui$estlp,NonEqui$selp)
```

```
Number of pointwise coverages: 951
Number of simultaneous coverages: 14
```

Smoothing Spline:

```
> coverage(x,NonEqui$estss,NonEqui$sess)
```

```
Number of pointwise coverages: 874
Number of simultaneous coverages: 104
```

The coverage rates of the pointwise confidence intervals at $x = 0.5$ are now 666/1000 for Nadaraya-Watson, 951/1000 for Local Polynomial and 874/1000 for Smoothing Splines. The corresponding simultaneous coverage rates are 0/1000, 14/1000 and 104/1000.