Solution to Series 10

1. **a)** Starting from Equation (1), we can multiply -1 and add y_i to both sides giving:

$$y_i - \hat{m}^{(-i)}(x_i) = y_i - \frac{(Sy)_i - S_{ii}y_i}{1 - S_{ii}},$$

Using $(Sy)_i = \hat{m}(x_i)$, and rewriting yields

$$y_i - \hat{m}^{(-i)}(x_i) = \frac{y_i(1 - S_{ii}) - \hat{m}(x_i) + S_{ii}y_i}{1 - S_{ii}} = \frac{y_i - \hat{m}(x_i)}{1 - S_{ii}}.$$

We finally obtain the formula for LOOCV error by taking squares on both sides and average over $i = 1, \ldots, n$

b) First let us consider the dataset without the i^{th} observation (x_i, y_i). The corresponding *hat* matrix, denoted $H^{(-i)}$, is of size $(n-1) \times (n-1)$ and is obtained from the formula

$$H^{(-i)} = X^{(-i)} \left((X^{(-i)})^T X^{(-i)} \right)^{-1} (X^{(-i)})^T,$$

where $X^{(-i)}$ is the design matrix without the i^{th} row (observation). Let us replace this i^{th} observation by the data point $(x_i, \hat{y}_i^{(-i)})$, where $\hat{y}_i^{(-i)}$ denotes the fitted value at x_i from the model without the i^{th} observation (i.e. with the fit corresponding to the *hat* matrix $H^{(-i)}$). We denote by z (as in the hint) the new response obtained from y by substituting y_i with the fitted value $\hat{y}_i^{(-i)}$. Since the replaced observation keeps the same predictor x_i, the new *hat* matrix is once again H. Moreover, as the fitted value already lies on the regression space obtained from the fitted model without the i^{th} observation, we can show that $(Hz)_i = \hat{y}_i^{(-i)}$. Indeed, the following inequality holds (where x_j^T denotes the j^{th} row of the matrix X):

$$\begin{aligned} \min_\beta \sum_{k=1}^n (z_k - x_k^T\beta)^2 = \min_\beta \left[\sum_{k \neq i}^n (y_k - x_k^T\beta)^2 + (\hat{y}_i^{(-i)} - x_i^T\beta)^2 \right] \\ \geq \min_\beta \left[\sum_{k \neq i}^n (y_k - x_k^T\beta)^2 + 0 \right]. \end{aligned}$$

This means that the OLS solution $\hat{\beta}^{(-i)}$ to the second regression problem (the fit on the dataset without the i^{th} observation on the right of the inequality) is also the optimal solution to the first OLS problem (the regression of z on X on the left of the inequality) by uniqueness of the minimizer. In particular, the fitted values for observation i are the same for both models, i.e $(Hz)_i = \hat{y}_i^{(-i)}$. We can further develop this equality:

$$\begin{aligned} \hat{y}_i^{(-i)} &= (Hy)_i = \sum_{j:j \neq i} H_{ij}y_j + H_{ii}\hat{y}^{(-i)} \\ &= \sum_j H_{ij}y_j - H_{ii}y_i + H_{ii}\hat{y}^{(-i)}, \end{aligned}$$

a reordering of the last relation gives us:

$$\hat{y}_i^{(-i)} = \frac{(Hy)_i - H_{ii}y_i}{1 - H_{ii}},$$

which shows that Equation (1) for linear regression (meaning that the formula for LOOCV error is valid for linear regression).

2. **a)** Thanks to the formula for the volume of a ball in $\mathbb{R}^n$, we can express $P(\|X_{(1)}\|_2 > t)$ for $t \in [0,1]$ as follows:

$$P(\|X_{(1)}\|_2 > t) = (1 - P(\|X_1\|_2 \leq t))^n = (1 - t^n)^n.$$

This last relation is obtained by observing that the event $\{\|X_{(1)}\|_2 > t\}$ is indeed the same event as $\{\|X_i\|_2 > t$ for all $i = 1, \ldots, n\}$ and using that the observations are all i.i.d. The value t^n is obtained by taking the ratio between the volume of a ball of radius t and the volume of a ball of radius 1. Indeed thanks to the formula given as an hint and because the distribution is uniformly distributed on the unit ball we have:

$$P(\|X_1\|_2 \leq t) = \frac{V_n(t)}{V_n(1)} = t^n.$$

b) Since the distribution of $\|X_{(1)}\|_2$ is continuous, the median t_{med} is defined as the solution of the equation

$$P(\|X_{(1)}\|_2 > t_{med}) = (1 - t_{med}^n)^n = \frac{1}{2}.$$

This gives us

$$t_{med} = \left(1 - (1/2)^{1/n}\right)^{1/n}.$$

c) We define the function found in the last point for the median and evaluate it for $n = 500$:

```
> f <- function(n) {(1-(1/2)^{1/n})^{1/n}}
> f(n = 500)
[1] 0.9869226
```

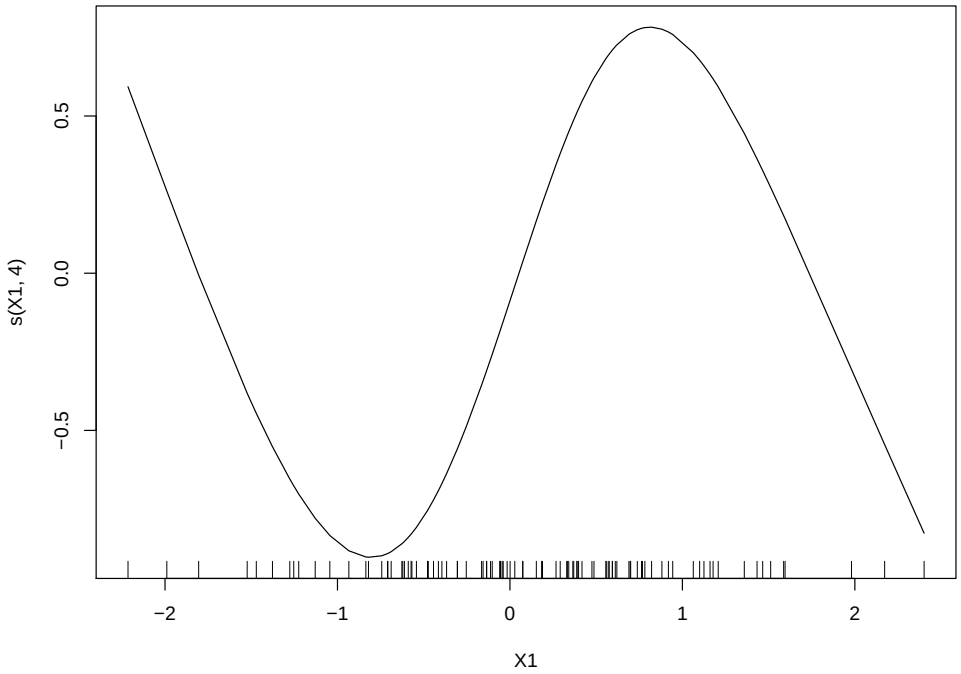
This indicates that the points concentrate on the sphere when the number of dimensions increases. This means for example that if we have in mind to estimate a regression function near 0, then few points will be sufficiently close to it in high-dimensions.

3. **a)** Nothing to do there apart from installing the required packages and creating the datasets.

b) Here is the code to perform the required fits:

```
> # store the mean squared error for the 12 models
> mse <- numeric(12)
> # store the names of the variables
> list.var <- names(dtrain)[1]
> # define the formulas
> form.lm <- as.formula(paste("y~",
                  paste(list.var, collapse = "+"), sep = ""))
> form.gam <- as.formula(paste("y~s(",
                  paste(list.var, collapse = ", 4)+s(",",4)", sep = ""))
> # fit the models (GAM + LM)
> gm <- gam(form.gam, data = dtrain)
> l <- lm(form.lm, data = dtrain)
> # record the mean squared errors
> mse[1] <- mean((dtest$y-predict(gm, dtest))^2)
> mse[2] <- mean((dtest$y-predict(l, dtest))^2)
> # fit he knn's with the function knn.reg
> for(k in 1:10) {
   kn <- knn.reg(train = matrix(dtrain[,1],ncol=1),
                 y = ytrain, test = matrix(dtest[,1],ncol=1), k = k)
   mse[2+k] <-  mean((dtest$y-kn$pred)^2)
 }
> # print the mse error
> print(mse)
```

```
 [1] 0.1385982 0.6329264 0.2025036 0.1328669 0.1115425
 [6] 0.1110039 0.1128088 0.1207373 0.1278271 0.1304200
[11] 0.1397241 0.1523280
> # plot of the fitted spline
> plot(gm)
```



We observe that linear regression gives the worse prediction error, which was expected due to the non-linearity of the signal. KNN succeeds in capturing the non-linearity of the signal due to its non-parametric and flexible nature. The GAM succeeds also in capturing the non-linearity. Moreover it furnishes a smooth non-linear fit as we can see on the plot.

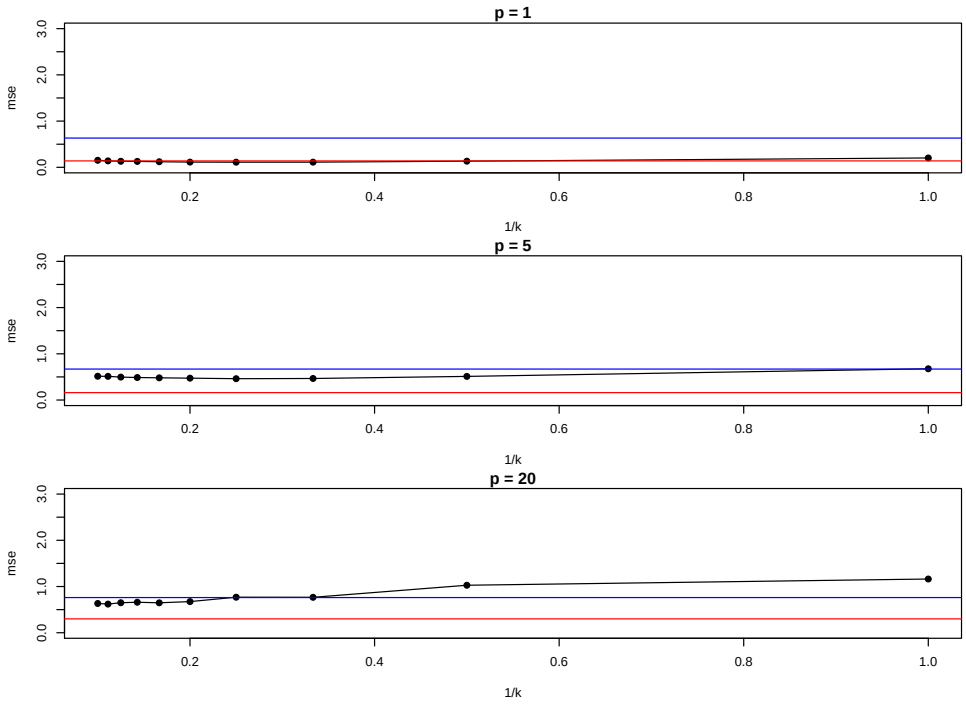
c) Here is the code to perform the required fits:

```
> # store the mse test error
> mse <- matrix(nrow=12,ncol=20)
> # iterate over the different numbers of additional variables
> # (please note we include also) the case without additional variables
> # here
> for (i in 1:20) {

 # names of the variables
 list.var <- names(dtrain)[1:i]

 # formulas
 form.lm <- as.formula(paste("y~",
                       paste(list.var, collapse = "+"), sep = ""))
 form.gam <- as.formula(paste("y~s(",
                        paste(list.var, collapse = ", 4)+s(",",4)", sep = ""))

 # GAM + LM fits
 gm <- gam(form.gam, data = dtrain)
 l <- lm(form.lm, data = dtrain)

 # test mse for both models
 mse[1,i] <- mean((dtest$y-predict(gm, dtest))^2)
 mse[2,i] <- mean((dtest$y-predict(l, dtest))^2)

 # fitting all knn's
 for(k in 1:10) {
```

```
  if(i == 1) {
    kn <- knn.reg(train = matrix(dtrain[,1:i],ncol=1),
                  y = ytrain, test = matrix(dtest[,1:i],ncol=1), k = k)
  } else {
    kn <- knn.reg(train = dtrain[,1:i],y = ytrain,
                  test = dtest[,1:i], k = k)
  }
  mse[2+k,i] <-  mean((dtest$y-kn$pred)^2)
}

}
```

d) Let us plot the test mean-squared errors for the three models: KNN (black), GAM (red) and linear regression (blue):

```
> par(mfrow=c(3,1))
> z <- 1
> plot(1/c(1:10), mse[3:12,z], ylim = c(0, 3),type="l",
      main = "p = 1",xlab="1/k",ylab="mse")
> points(1/c(1:10), mse[3:12,z],pch=19)
> abline(h = mse[1,z],col="red")
> abline(h = mse[2,z],col="blue")
> z <- 5
> plot(1/c(1:10), mse[3:12,z], ylim = c(0, 3),type="l",
      main = "p = 5",xlab="1/k",ylab="mse")
> points(1/c(1:10), mse[3:12,z],pch=19)
> abline(h = mse[1,z],col="red")
> abline(h = mse[2,z],col="blue")
> z <- 20
> plot(1/c(1:10), mse[3:12,z], ylim = c(0, 3),type="l",
      main = "p = 20",xlab="1/k",ylab="mse")
> points(1/c(1:10), mse[3:12,z],pch=19)
> abline(h = mse[1,z],col="red")
> abline(h = mse[2,z],col="blue")
```



As we can see from the plots, when we increase the number of dimensions, KNN gets worse due to the curse of dimensionality, whereas the GAM model does not suffer too much from this issue. Note also that the linear regression does not seem to suffer too much from the curse of dimensionality, even if its test error is big due to the non-linearity of the signal.

4. **a)** Let us run the backfitting algorithm for the first dataset

```
> ##### Backfitting, using functions g_j(x) = alpha_j + beta_j(x)
>
> # initialization
> mu.hat <- mean(y)
> g1.old = g1.new = rep(0, n)  # initial values of g1.hat, etc
> g2.old = g2.new = rep(0, n)
> g3.old = g3.new = rep(0, n)
> g4.old = g4.new = rep(0, n)
> # loop until convergence criterion satisfied
> eps <- 10^(-10)
> conv <- 0
> iter <- 0
> while(!conv){

   # for x1
   r <- y - mu.hat - g2.new - g3.new - g4.new
   fit1 <- lm(r~x1)
   g1.new <- fit1$fitted

   # for x2
   r <- y - mu.hat - g1.new - g3.new - g4.new
   fit2 <- lm(r~x2)
   g2.new <- fit2$fitted

   # for x3
   r <- y - mu.hat - g1.new - g2.new - g4.new
   fit3 <- lm(r~x3)
   g3.new <- fit3$fitted

   # for x4
   r <- y - mu.hat - g1.new - g2.new - g3.new
   fit4 <- lm(r~x4)
   g4.new <- fit4$fitted

   # check convergence:
   conv <- max( sum( (g1.new-g1.old)^2 ) / sum(g1.old^2),
                sum( (g2.new-g2.old)^2 ) / sum(g2.old^2),
                sum( (g3.new-g3.old)^2 ) / sum(g3.old^2),
                sum( (g4.new-g4.old)^2 ) / sum(g4.old^2) ) < eps

   # update iter and gj.old:
   iter <- iter+1
   g1.old <- g1.new
   g2.old <- g2.new
   g3.old <- g3.new
   g4.old <- g4.new
 }
> iter  # number of iterations
[1] 5
> beta0.hat <- mu.hat + fit1$coef[1] + fit2$coef[1] + fit3$coef[1] + fit4$coef[1]
> (beta.hat <- c(beta0.hat,fit1$coef[2],fit2$coef[2],fit3$coef[2],fit4$coef[2]))
(Intercept)          x1          x2          x3          x4
  0.6430246   1.1933286   2.1696399   0.5337476  -2.9573178
> # compare with multiple regression
> lm(y~x1+x2+x3+x4)$coef
(Intercept)          x1          x2          x3          x4
  0.6430246   1.1933287   2.1696399   0.5337476  -2.9573178
```

```
>
```

now we change the order of variables over which we cycle:

```
> ##### Backfitting, using functions g_j(x) = alpha_j + beta_j(x)
>
> # initialization
> mu.hat <- mean(y)
> g1.old = g1.new = rep(0, n)  # initial values of g1.hat, etc
> g2.old = g2.new = rep(0, n)
> g3.old = g3.new = rep(0, n)
> g4.old = g4.new = rep(0, n)
> # loop until convergence criterion satisfied
> eps <- 10^(-10)
> conv <- 0
> iter <- 0
> while(!conv){

   # for x3
   r <- y - mu.hat - g1.new - g2.new - g4.new
   fit3 <- lm(r~x3)
   g3.new <- fit3$fitted

   # for x2
   r <- y - mu.hat - g1.new - g3.new - g4.new
   fit2 <- lm(r~x2)
   g2.new <- fit2$fitted

   # for x4
   r <- y - mu.hat - g1.new - g2.new - g3.new
   fit4 <- lm(r~x4)
   g4.new <- fit4$fitted

   # for x1
   r <- y - mu.hat - g2.new - g3.new - g4.new
   fit1 <- lm(r~x1)
   g1.new <- fit1$fitted


   # check convergence:
   conv <- max( sum( (g1.new-g1.old)^2 ) / sum(g1.old^2),
                sum( (g2.new-g2.old)^2 ) / sum(g2.old^2),
                sum( (g3.new-g3.old)^2 ) / sum(g3.old^2),
                sum( (g4.new-g4.old)^2 ) / sum(g4.old^2) ) < eps

   # update iter and gj.old:
   iter <- iter+1
   g1.old <- g1.new
   g2.old <- g2.new
   g3.old <- g3.new
   g4.old <- g4.new
 }
> iter  # number of iterations
[1] 5
> # all intercepts
> beta0.hat <- mu.hat + fit1$coef[1] + fit2$coef[1] + fit3$coef[1] + fit4$coef[1]
> (beta.hat <- c(beta0.hat,fit1$coef[2],fit2$coef[2],fit3$coef[2],fit4$coef[2]))
(Intercept)          x1          x2          x3          x4
  0.6430246   1.1933287   2.1696399   0.5337476  -2.9573178
```

```
> # compare with multiple regression
> lm(y~x1+x2+x3+x4)$coef
```

```
(Intercept)          x1          x2          x3          x4
  0.6430246   1.1933287   2.1696399   0.5337476  -2.9573178
```

Note that the backfitting estimates are really close to the regression estimates. We now apply the backfitting algorithm for the second dataset, where correlation exists between the predictors for two different cycling orders:

```
> ##### Backfitting, using functions g_j(x) = alpha_j + beta_j(x)
>
> # initialization
> mu.hat <- mean(y_cor)
> g1.old = g1.new = rep(0, n)  # initial values of g1.hat, etc
> g2.old = g2.new = rep(0, n)
> g3.old = g3.new = rep(0, n)
> g4.old = g4.new = rep(0, n)
> # loop until convergence criterion satisfied
> eps <- 10^(-10)
> conv <- 0
> iter <- 0
> while(!conv){

   # for x1
   r <- y_cor - mu.hat - g2.new - g3.new - g4.new
   fit1 <- lm(r~x1_cor)
   g1.new <- fit1$fitted

   # for x3
   r <- y_cor - mu.hat - g1.new - g2.new - g4.new
   fit3 <- lm(r~x3_cor)
   g3.new <- fit3$fitted

   # for x2
   r <- y_cor - mu.hat - g1.new - g3.new - g4.new
   fit2 <- lm(r~x2_cor)
   g2.new <- fit2$fitted

   # for x4
   r <- y_cor - mu.hat - g1.new - g2.new - g3.new
   fit4 <- lm(r~x4_cor)
   g4.new <- fit4$fitted

   # check convergence:
   conv <- max( sum( (g1.new-g1.old)^2 ) / sum(g1.old^2),
                sum( (g2.new-g2.old)^2 ) / sum(g2.old^2),
                sum( (g3.new-g3.old)^2 ) / sum(g3.old^2),
                sum( (g4.new-g4.old)^2 ) / sum(g4.old^2) ) < eps

   # update iter and gj.old:
   iter <- iter+1
   g1.old <- g1.new
   g2.old <- g2.new
   g3.old <- g3.new
   g4.old <- g4.new
 }
> iter  # number of iterations
```

```
[1] 75
```

```
> beta0.hat <- mu.hat + fit1$coef[1] + fit2$coef[1] + fit3$coef[1] + fit4$coef[1]
> (beta.hat <- c(beta0.hat,fit1$coef[2],fit2$coef[2],fit3$coef[2],fit4$coef[2]))
```

```
(Intercept)      x1_cor      x2_cor      x3_cor      x4_cor
-0.01980102  0.38721927  1.91144995  0.57536342 -2.47870082
> # compare with multiple regression
> lm(y_cor~x1_cor+x2_cor+x3_cor+x4_cor)$coef
(Intercept)      x1_cor      x2_cor      x3_cor      x4_cor
-0.01980078  0.38723875  1.91144997  0.57536334 -2.47871636
>
```

now we change the order of variables over which we cycle:

```
> ##### Backfitting, using functions g_j(x) = alpha_j + beta_j(x)
>
> # initialization
> mu.hat <- mean(y_cor)
> g1.old = g1.new = rep(0, n)  # initial values of g1.hat, etc
> g2.old = g2.new = rep(0, n)
> g3.old = g3.new = rep(0, n)
> g4.old = g4.new = rep(0, n)
> # loop until convergence criterion satisfied
> eps <- 10^(-10)
> conv <- 0
> iter <- 0
> while(!conv){

   # for x3
   r <- y_cor - mu.hat - g1.new - g2.new - g4.new
   fit3 <- lm(r~x3_cor)
   g3.new <- fit3$fitted

   # for x2
   r <- y_cor - mu.hat - g1.new - g3.new - g4.new
   fit2 <- lm(r~x2_cor)
   g2.new <- fit2$fitted

   # for x4
   r <- y_cor - mu.hat - g1.new - g2.new - g3.new
   fit4 <- lm(r~x4_cor)
   g4.new <- fit4$fitted

   # for x1
   r <- y_cor - mu.hat - g2.new - g3.new - g4.new
   fit1 <- lm(r~x1_cor)
   g1.new <- fit1$fitted


   # check convergence:
   conv <- max( sum( (g1.new-g1.old)^2 ) / sum(g1.old^2),
                sum( (g2.new-g2.old)^2 ) / sum(g2.old^2),
                sum( (g3.new-g3.old)^2 ) / sum(g3.old^2),
                sum( (g4.new-g4.old)^2 ) / sum(g4.old^2) ) < eps

   # update iter and gj.old:
   iter <- iter+1
   g1.old <- g1.new
   g2.old <- g2.new
   g3.old <- g3.new
   g4.old <- g4.new
 }
> iter  # number of iterations
[1] 61
```

```
> # all intercepts
> beta0.hat <- mu.hat + fit1$coef[1] + fit2$coef[1] + fit3$coef[1] + fit4$coef[1]
> (beta.hat <- c(beta0.hat,fit1$coef[2],fit2$coef[2],fit3$coef[2],fit4$coef[2]))
(Intercept)      x1_cor      x2_cor      x3_cor      x4_cor
-0.01980092  0.38721734  1.91144994  0.57536345 -2.47869632
> # compare with multiple regression
> lm(y_cor~x1_cor+x2_cor+x3_cor+x4_cor)$coef
(Intercept)      x1_cor      x2_cor      x3_cor      x4_cor
-0.01980078  0.38723875  1.91144997  0.57536334 -2.47871636
```

We remark that even when the predictors are correlated, the backfitting approach still gives reasonable estimates and does not seem to depend on the order of the variables in the cycle.