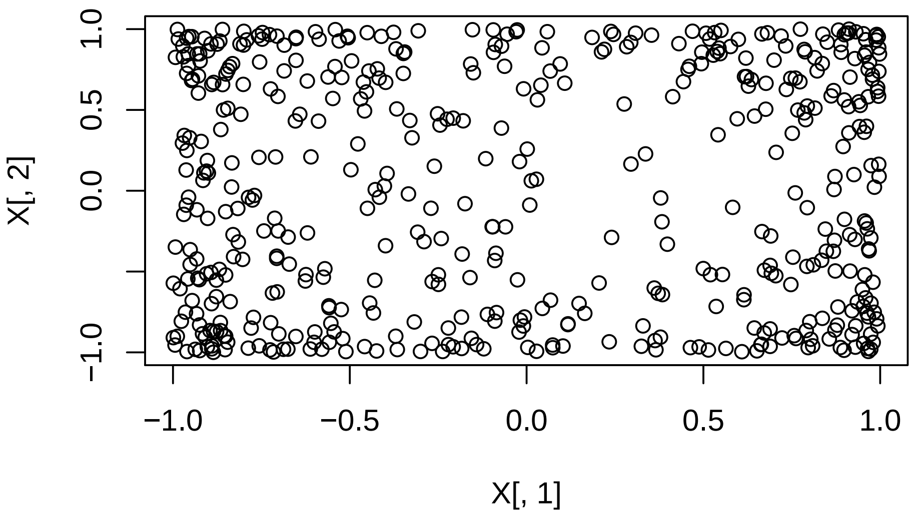
Solution to Series 4

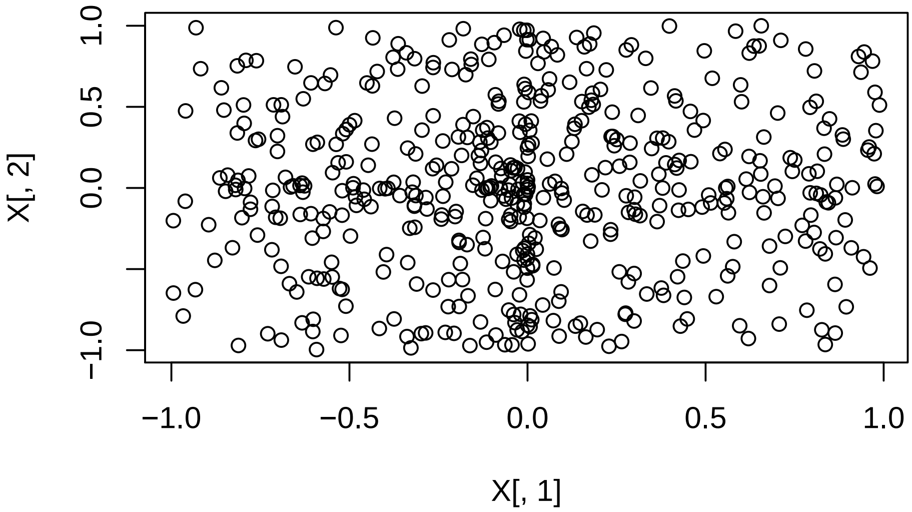
1. a) We simulate X and plot its columns for $p = 2$ as follows.

```
> library("kknn") #install.packages("kknn")
> set.seed(0)
> g1<-function(x){2*x/(1+abs(x)^1.5)} # Favour x-values with larger absolute value
> g2<-function(x){x^3/abs(x)^1.5} # Favour x-values with smaller absolute value
> g3<-function(x){x} # Keep the uniform distribution
> g<-g1  #This is our choice
> n<-500 #number of observations that we have available for CV
> p=2          #number of predictors
> z<-runif(n*p,min=-1,max=1)
> Z<-matrix(z,ncol=p)
> X <- g1(Z)
> plot(X[,1],X[,2])
```



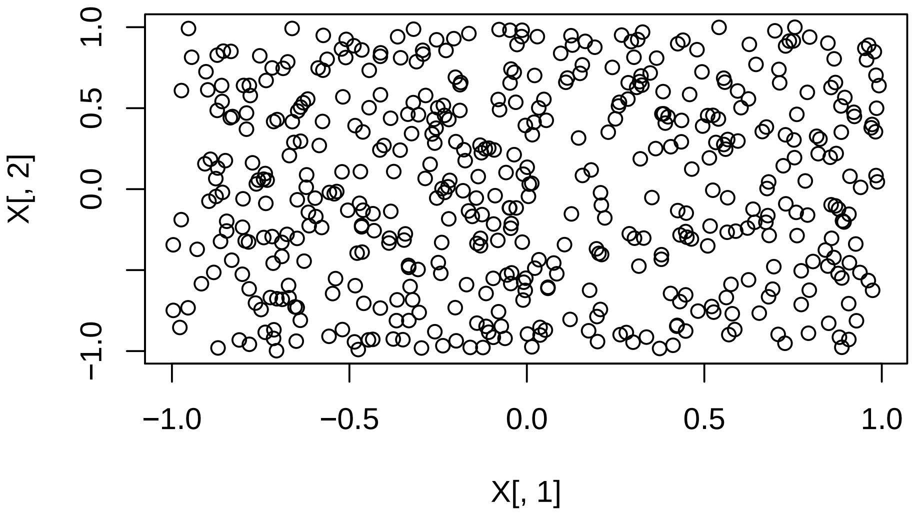
The above plot is for $g =$ g1. For $g =$ g2, a corresponding set of samples looks like this (the samples are more frequent around 0):

```
> X <- g2(Z)
> plot(X[,1],X[,2])
```



For $g =$ g3, we have the uniform distribution.

```
> X <- g3(Z)
> plot(X[,1],X[,2])
```



b) We choose $g =$ g1. The function `sampleX()` can be defined as:

```
> sampleX<-function(n=500){
          z=runif(n*p,min=-1,max=1)
          Z=matrix(z,ncol=p)
          X<-g1(Z)
          return(X);
  }
```

c) A possible definition of `f` is this:

```
> f1dim<-function(x){ sin(8*x)/(1+(4*x)^2) }
> f<-function(X){
    return(f1dim(X[,1]));
  }
```

d) With the following function, we can generate samples for Y given the samples in `X` according to the specified model. We chose 0.3 for the standard deviation of the noise ε.

```
> sampleY<-function(X){
          return(f(X)+rnorm(dim(X)[1],sd=0.3))
  }
```

e) The true test MSE can be approximated with simulation as follows:

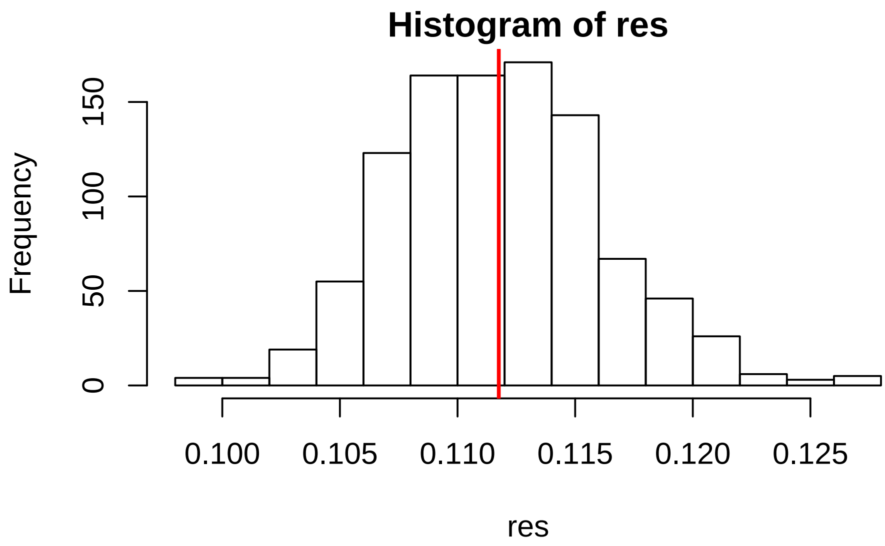
```
> iterations<-1000
> res <- numeric(iterations)
> for (i in 1:iterations) {
          Xtrain<-sampleX()
          Ytrain<-sampleY(Xtrain)
          dfTrain=data.frame(y=Ytrain,x=Xtrain)
          Xtest<-sampleX(2000)
          Ytest<-sampleY(Xtest)
          dfTest=data.frame(x=Xtest)

          fit.kknn <- kknn(y ~ ., dfTrain,dfTest,k=8)
          predTest=predict(fit.kknn)

          # This approximates the expected test mse for the trained predictor
          res[i] <- mean((predTest-Ytest)^2)
  }
> EstimateTrueTestMSE <- mean(res)
> EstimateTrueTestMSE
[1] 0.1117533
```

We can then use a histogram to visualize the distribution.

```
> hist(res)
> abline(v=EstimateTrueTestMSE, col="red",lwd=2)
```



f) We define the functions for the estimation of the expected test MSE using the five specified methods:

```
> X<-sampleX()
> Y<-sampleY(X)
> # Validation set approach
>
> ValidationSet<-function(X,Y){
        n<-length(Y)
        s <- sample(1:n, size=n, replace=F)
        folds <- cut(seq(1,n), breaks=2, labels=FALSE)
        ind.test <- s[which(folds==1)]
        dfTrain=data.frame(y=Y[ind.test],x=X[ind.test,])
        dfTest=data.frame(x=X[-ind.test,])
        fit.kknn <- kknn(y ~ ., dfTrain,dfTest,k=8)
        predTest=predict(fit.kknn)
        Ytest<-Y[-ind.test]
        MSEEstimate=mean((predTest-Ytest)^2)
        return(MSEEstimate)
 }
> ValidationSet(X,Y) #estimate for the given X and Y
[1] 0.1185164
> # Repeated Validation set approach (10 times average)
>
> RepeatedValidationSet<-function(X,Y){
        MSEEstimate <- replicate(10, ValidationSet(X,Y))
        return(mean(MSEEstimate))
 }
> RepeatedValidationSet(X,Y)
[1] 0.1230888
> # 10 Fold CV
>
> TenFoldCV <- function(X,Y){
        MSEEstimateFolds <- numeric(10)
        n <- length(Y)
        s <- sample(1:n, size=n, replace=F)
        folds <- cut(seq(1,n), breaks=10, labels=FALSE)
        for (i in 1:10) {
                ind.test <- s[which(folds==i)]
```

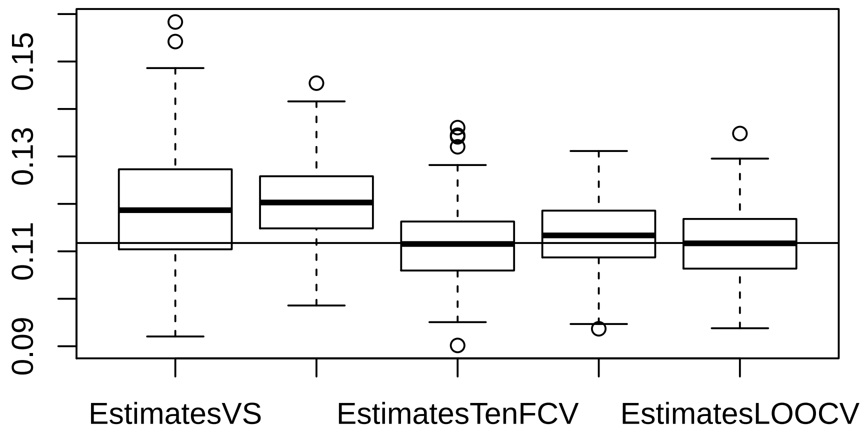
```
                dfTrain=data.frame(y=Y[-ind.test],x=X[-ind.test,])
                dfTest=data.frame(x=X[ind.test,])
                fit.kknn <- kknn(y ~ ., dfTrain,dfTest,k=8)  # k=8
                predTest=predict(fit.kknn)
                Ytest <- Y[ind.test]
                MSEEstimateFolds[i] <- mean((predTest-Ytest)^2)
        }
        return(mean(MSEEstimateFolds))
 }
> TenFoldCV(X,Y)
[1] 0.1110407
> # Repeated 10 Fold CV (10 times average)
>
> RepeatedTenFoldCV<-function(X,Y){
   MSEEstimate <- replicate(10, TenFoldCV(X,Y))
   return(mean(MSEEstimate))
 }
> RepeatedTenFoldCV(X,Y)
[1] 0.1122765
> # Leave-one-out CV
>
> LOOCV<-function(X,Y){
        n <- length(Y)
        MSEEstimate <- numeric(n)
        for (i in 1:n) {
                dfTrain=data.frame(y=Y[-i],x=X[-i,])
                dfTest=data.frame(x=matrix(X[i,],nrow=1))
                fit.kknn <- kknn(y ~ ., dfTrain,dfTest,k=8)
                predTest=predict(fit.kknn)
                Ytest<-Y[i]
                MSEEstimate[i]<-(predTest-Ytest)^2
        }
        return(mean(MSEEstimate))
 }
> LOOCV(X,Y)
[1] 0.1111346
>
```

Note that we explicitly need explicitly convert `X[i,]` to a matrix `dfTest=data.frame(x=matrix(X[i,],nrow=1))` in the implementation for `LOOCV` because we only selected one row which would otherwise not result in a matrix.

g) We use the provided function `EvaluateOnSimulation` to generate samples of the estimates of the five different methods for expected test MSE estimation. Then we use a boxplot to visualize the results.

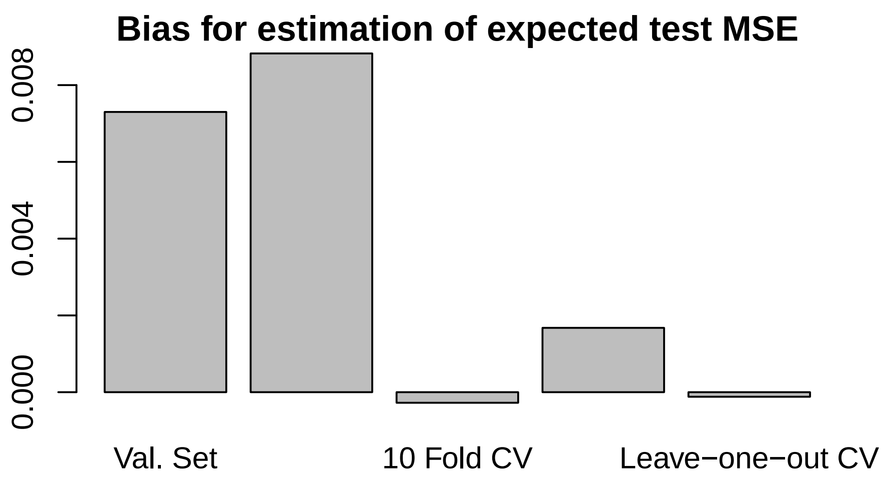
```
> EvaluateOnSimulation<-function(estimationFunction, iterations=200){
        result<-numeric(iterations)
        for (i in 1:iterations) {
                X<-sampleX()
                Y<-sampleY(X)
                result[i]= estimationFunction(X,Y)
        }
        return(result)
 }
> EstimatesVS <- EvaluateOnSimulation(ValidationSet)
> EstimatesRVS <- EvaluateOnSimulation(RepeatedValidationSet)
> EstimatesTenFCV <- EvaluateOnSimulation(TenFoldCV)
> EstimatesRTenFCV <- EvaluateOnSimulation(RepeatedTenFoldCV)
> EstimatesLOOCV <- EvaluateOnSimulation(LOOCV)
```

```
> # look at results, for example create boxplot:
> Estimates <- cbind(EstimatesVS,EstimatesRVS,EstimatesTenFCV,EstimatesRTenFCV,EstimatesLOOCV)
> boxplot(Estimates)
> abline(h=EstimateTrueTestMSE)
```



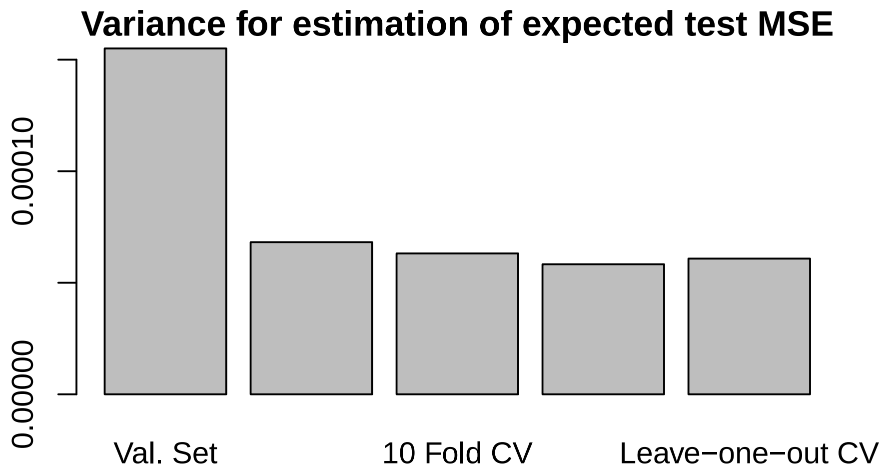
h) Using the results from the previous subtask, calculate approximations for bias and variance. The bias approximation uses the approximation of the true expected test MSE from task e).

```
> # Bias
> biasVS <- mean(EstimatesVS) - EstimateTrueTestMSE
> biasRVS <- mean(EstimatesRVS) - EstimateTrueTestMSE
> biasTenFCV <- mean(EstimatesTenFCV) - EstimateTrueTestMSE
> biasRTenFCV <- mean(EstimatesRTenFCV) - EstimateTrueTestMSE
> biasLOOCV <- mean(EstimatesLOOCV) - EstimateTrueTestMSE
> # Variance
> varVS <- var(EstimatesVS)
> varRVS <- var(EstimatesRVS)
> varTenFCV <- var(EstimatesTenFCV)
> varRTenFCV <- var(EstimatesRTenFCV)
> varLOOCV <- var(EstimatesLOOCV)
> caption<-c("Val. Set", "Repeated Val. Set", "10 Fold CV", "Repeated 10 Fold CV","Leave-one-c
> biases<-c(biasVS,biasRVS,biasTenFCV,biasRTenFCV,biasLOOCV)
> names(biases)=caption
> barplot(biases, main="Bias for estimation of expected test MSE")
```

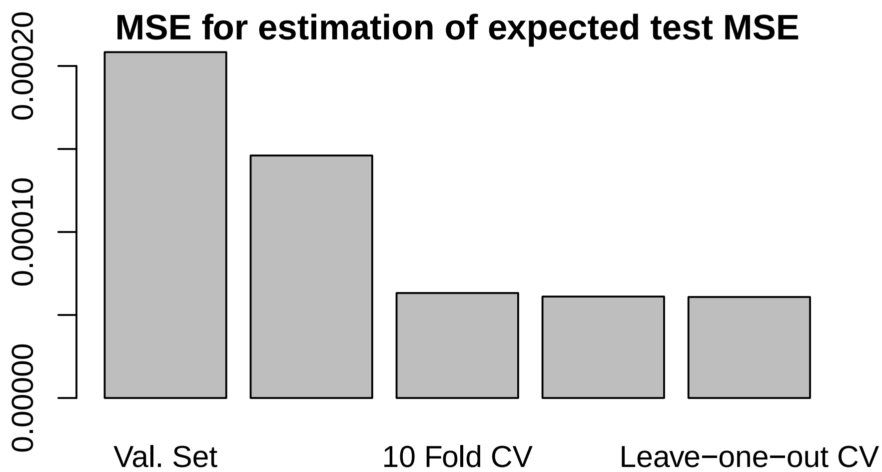



```
> variances<-c(varVS,varRVS,varTenFCV,varRTenFCV,varLOOCV)
```

```
> names(variances)=caption
> barplot(variances, main="Variance for estimation of expected test MSE")
```




```
> msemse<-biases^2+variances
> names(msemse)=caption
> barplot(msemse, main="MSE for estimation of expected test MSE")
```



i) The following function estimates the variance of the 10-Fold CV estimator using the provided formula.

```
> VarTenFoldCV <- function(X,Y){
   MSEEstimateFolds <- numeric(10)

   n <- length(Y)
   s <- sample(1:n, size=n, replace=F)
   folds <- cut(seq(1,n), breaks=10, labels=FALSE)

   for (i in 1:10) {
     ind.test <- s[which(folds==i)]

     dfTrain=data.frame(y=Y[-ind.test],x=X[-ind.test,])
     dfTest=data.frame(x=X[ind.test,])
     fit.kknn <- kknn(y ~ ., dfTrain,dfTest,k=8)  # k=8
     predTest=predict(fit.kknn)
     Ytest <- Y[ind.test]
     MSEEstimateFolds[i] <- mean((predTest-Ytest)^2)
   }
```

```
  return(var(MSEEstimateFolds)/10)
}
```