Exercise Set 1 - solutions and grading

Please read the following before doing the peer reviews:

- Problem sheet at https://moodle.helsinki.fi/pluginfile.php/3324196/mod_folder/content/0/DATA11002-2020-E1v2.pdf?forcedownload=1
- General grading instructions at https://moodle.helsinki.fi/mod/page/view.php?id=2072012
- Step-by-step instructions for the peer review week at https://moodle.helsinki.fi/mod/page/view.php?id=2084536

Problem 1

Task a

First 2 lines:

```
head -n 2 npf_train.csv
```

```
## "id","date","class4","partlybad","CO2168.mean","CO2168.std","CO2336.mean","CO2336.std","CO242.mean",
## 1,"2000-02-23","nonevent",FALSE,380.52811965812,0.802000657433635,380.371465517241,0.889550208090346
```

Last 2 lines:

```
tail -n 2 npf_train.csv
```

```
## 429,"2006-12-12","nonevent",FALSE,387.960192307692,0.40942566365253,388.0075,0.417033759859759,388.1
## 430,"2009-11-19","nonevent",FALSE,401.640476190476,2.05221157540603,400.852096774194,2.8841824918368
```

Task b

Counting lines:

```
wc -l npf_train.csv
```

```
##      431 npf_train.csv
```

I.e., the `npf_train.csv` file has 431 lines.

Task c

You can transform a csv file to a tsv csv file by `sed`:

```
sed "s/,/$(printf '\t')/g" npf_train.csv > npf_train.tsv
```

Alternatively, you can use `tr`:

```
tr "," "\t" < npf_train.csv > npf_train.tsv
```

Grading instructions: It is enough to use either `sed` or `tr`.

Task d

You can use `awk`, `sort`, and `uniq` to find unique classes in column 3:

```
awk -F ',' '{print $3}' npf_train.csv | sort | uniq
```

```
## "II"
## "Ia"
## "Ib"
## "class4"
## "nonevent"
```

If you want to rid of the column name "class4" that appears in the first line then you can use, e.g.:

```
tail -n $((`wc -l < npf_train.csv`-1)) npf_train.csv | awk -F ',' '{print $3}' | sort | uniq
```

```
## "II"
## "Ia"
## "Ib"
## "nonevent"
```

Grading instructions: The latter (filtering out "class4") is not required for full points.

Grading

Points: max. 5 (a:1, b:1, c:1, d: 2)

Problem 2

Task a

First load the dataset:

```
npf <- read.csv("npf_train.csv")
```

Task b

Then the following commands are run, as instructed:

```
View(npf)
```

```
rownames(npf) <- npf[,"date"]
```

```
npf <- npf[,-1]
```

We use here `View`, but you can use `fix` as well.

Grading: It is enough to report that the instructed commands were run in tasks a-b.

Task c

We will then use `summary`:

```
summary(npf)
```

```
##          date          class4    partlybad        CO2168.mean
##  2000-01-01:  1   Ia      : 26   Mode :logical   Min.   :360.5
##  2000-02-22:  1   Ib      : 83   FALSE:430       1st Qu.:373.2
##  2000-02-23:  1   II      :106                   Median :380.5
##  2000-03-25:  1   nonevent:215                   Mean   :381.4
##  2000-04-04:  1                                  3rd Qu.:388.2
##  2000-04-06:  1                                  Max.   :421.5
##  (Other)   :424
##    CO2168.std       CO2336.mean      CO2336.std        CO242.mean
##  Min.   : 0.1645   Min.   :360.4   Min.   : 0.1492   Min.   :361.8
##  1st Qu.: 0.8874   1st Qu.:373.3   1st Qu.: 0.8955   1st Qu.:374.5
```

```
##  Median : 2.2818   Median :380.5   Median : 2.1879   Median :381.4
##  Mean   : 3.2596   Mean   :381.4   Mean   : 3.0728   Mean   :382.4
##  3rd Qu.: 4.6052   3rd Qu.:388.2   3rd Qu.: 4.2729   3rd Qu.:388.7
##  Max.   :17.2848   Max.   :421.1   Max.   :15.9555   Max.   :422.6
##
##    CO242.std        CO2504.mean      CO2504.std        Glob.mean
##  Min.   : 0.1527   Min.   :360.0   Min.   : 0.1342   Min.   :  3.719
##  1st Qu.: 1.1535   1st Qu.:373.3   1st Qu.: 0.8906   1st Qu.: 74.046
##  Median : 2.6715   Median :380.5   Median : 2.0735   Median :200.062
...
```

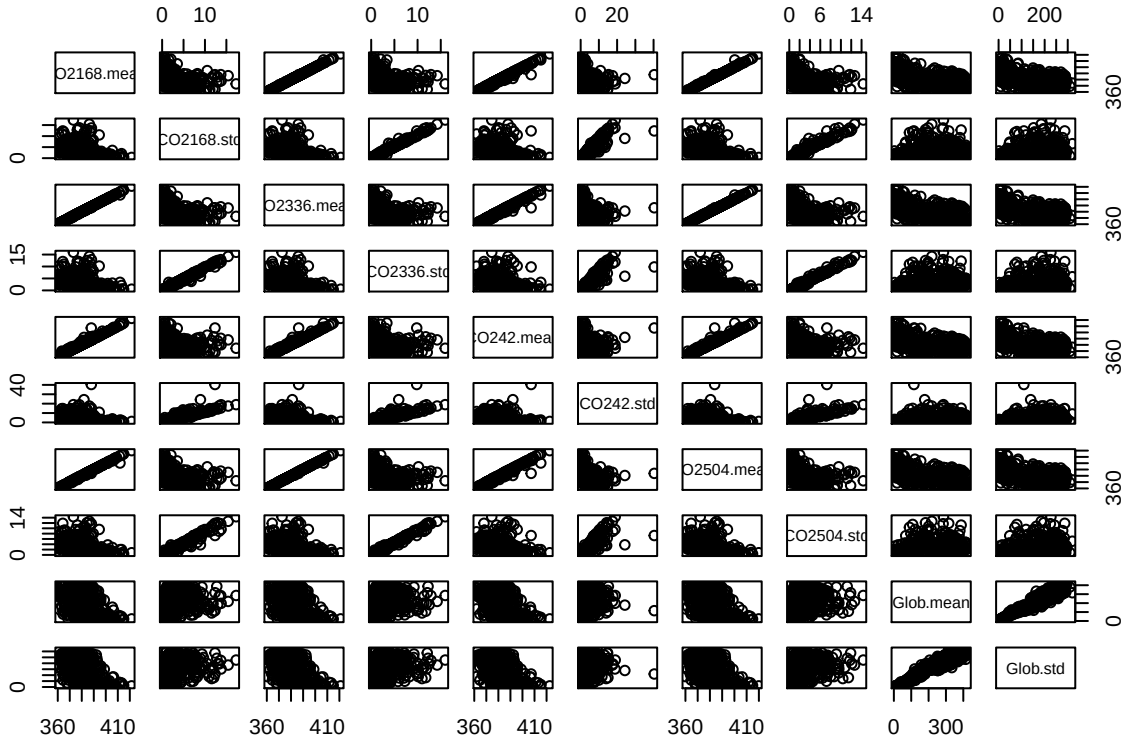
Grading: It is sufficient to output few lines or no lines at all.

Column 3 is removed, as instructed:

```
npf <- npf[,-3]
```

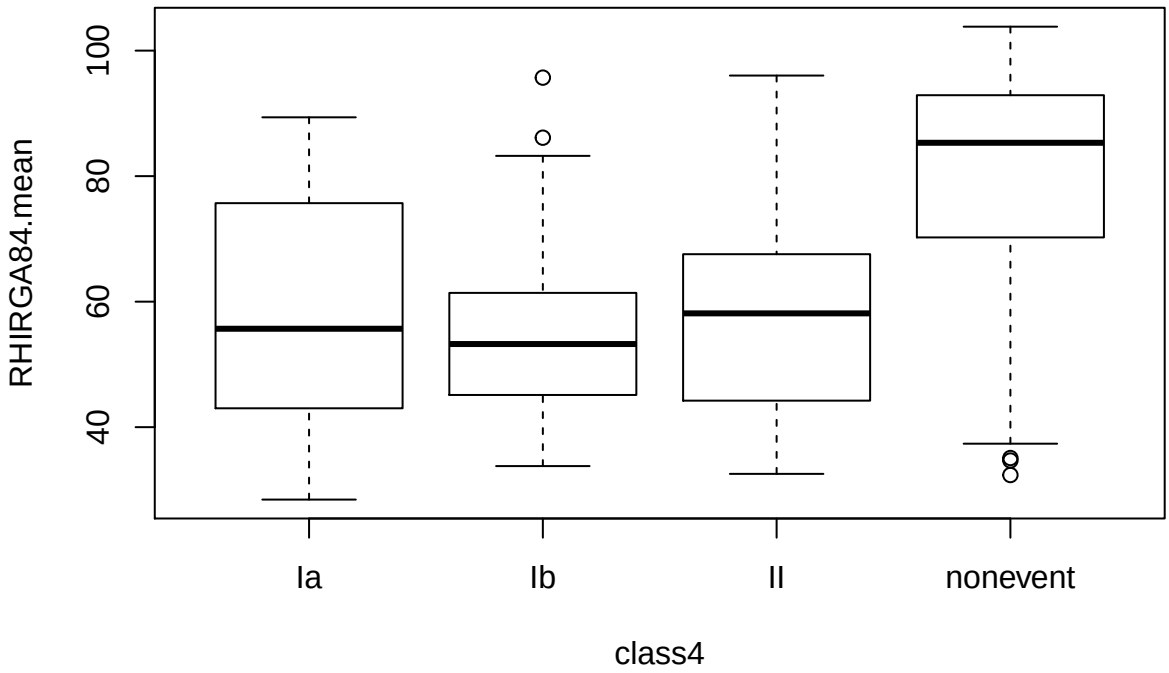
We'll then provide a pairplot of columns from 3 to 12:

```
pairs(npf[,3:12])
```



Then the requested boxplot:

```
boxplot(RHIRGA84.mean ~ class4,npf)
```



Lets create new class variable `class2`, as instructed:

```
npf$class2 <- factor("event",levels=c("nonevent","event"))
npf$class2[npf$class4=="nonevent"] <- "nonevent"
```

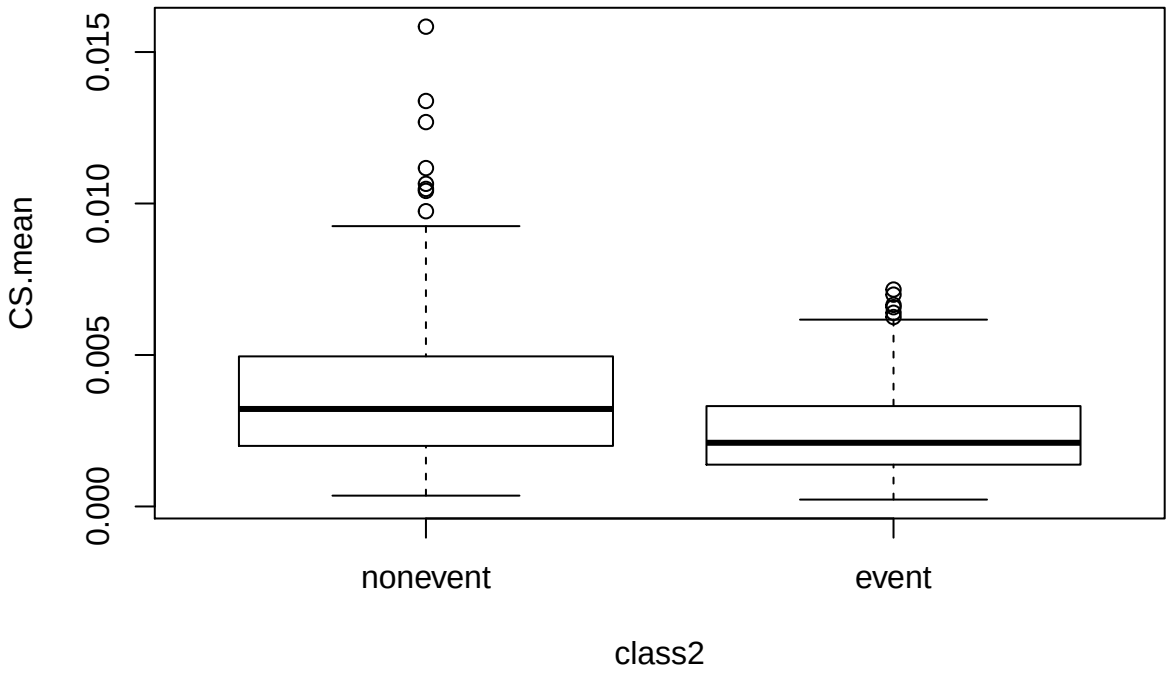
```
summary(npf$class2)
```

```
## nonevent    event
##      215      215
```

There are 215 nonevent days and equal number of event days.

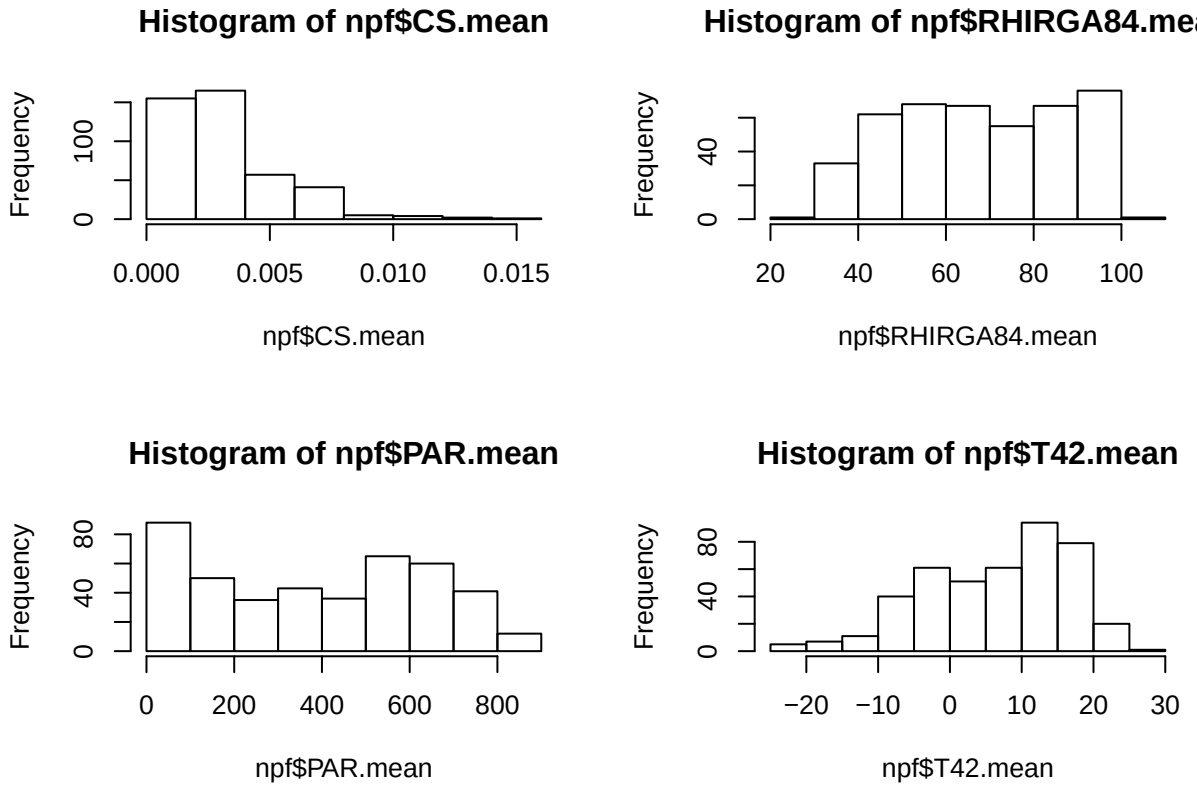
The requested boxplot:

```
boxplot(CS.mean ~ class2,npf)
```



Here are some requested histograms using the `hist` function, e.g.:

```
par(mfrow=c(2,2))
hist(npf$CS.mean)
hist(npf$RHIRGA84.mean)
hist(npf$PAR.mean)
hist(npf$T42.mean)
```



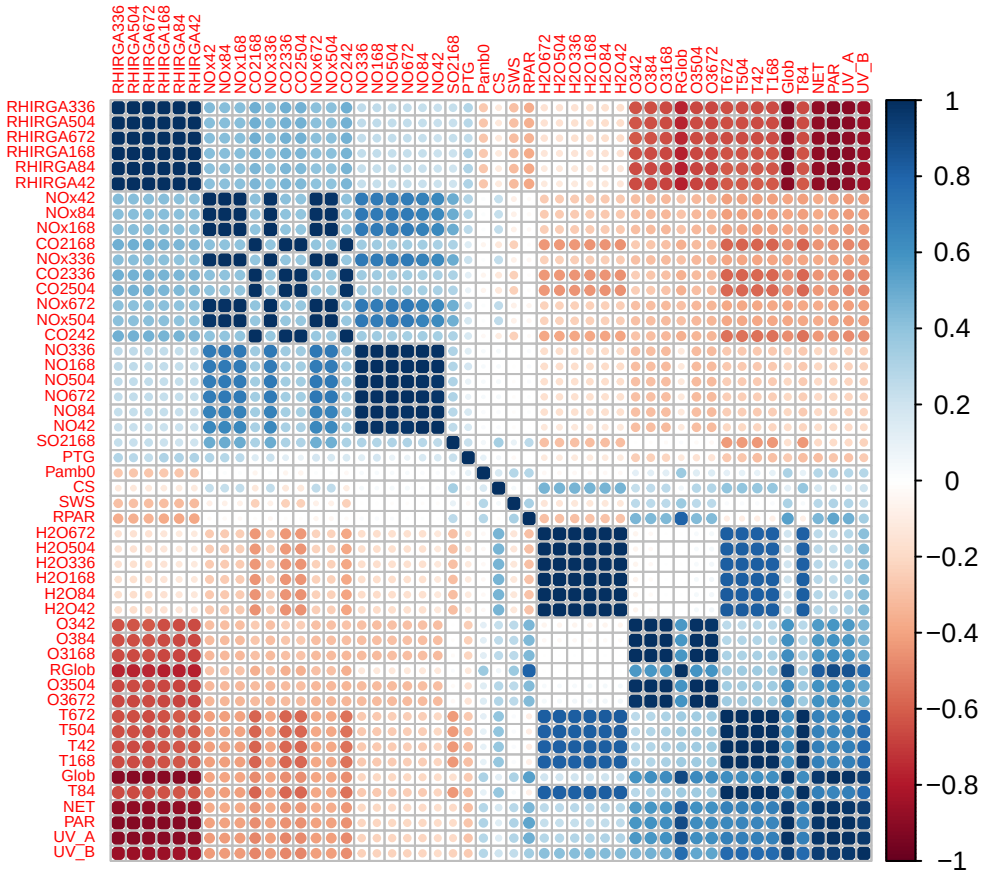
CS is condensation sink, RHIRGA84 is related to humidity, PAR is radiation, and T42 is temperature.

The student is asked to explore the data and provide a brief summary. The answer can contain anything, but I will do here a correlation plot of the mean values (variables ending with ".mean"):

```
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```
## column names of npf ending with ".mean"
i <- colnames(npf)[sapply(colnames(npf),
                          function(s) {
                            n <- nchar(s)
                            n>5 && substr(s,n-4,n)==".mean"
                          })]

corr <- cor(npf[,i])
## strip out the ".mean" from the row and column names
colnames(corr) <- rownames(corr) <- sapply(i,function(s) substr(s,1,nchar(s)-5))
## order the variables nicely by the 1st principal component
## (PCA later in the course!)
corrplot(corr,order="FPC",tl.cex=0.5)
```



We notice that there are several mutually correlated variable blocks. For example, relative humidity measurements (RHIRGA) at different heights are quite correlated, as various other measurements of same things at different heights.

Grading: The student gets full points from "explore the data and provide a brief summary" if at least one new item (not covered earlier) of the data is given (not necessarily the correlation plot shown above).

Grading

Points: max. 10 (a: 1, b: 1, c: 8)

In this Problem, the student can use R or Python or something else, as long as the students provides an answer to the questions asked and the answer is understandable to others (e.g., if a boxplot was asked, a boxplot is produced). If figure is requested it should be shown, otherwise it is sufficient that the answer states that the requested task has been done (e.g., loading the dataset). The important thing is that the dataset preprocessed as instructed (in a manner done in the programming environment) and the requested analysis is performed.

Problem 3

Task a

From $p_P = \prod_{i=1}^n p_i^{a_i}$ it follows that

$$l_P = \log p_P = \log \prod_{i=1}^n p_i^{a_i} = \sum_{i=1}^n a_i \log p_i = \sum_{i=1}^n a_i l_i,$$

which shows the first equation.

Second equation:

$$l_S = \log p_S = \log \sum_{i=1}^n p_i = \log m \times \sum_{i=1}^n p_i/m = \log m + \log \sum_{i=1}^n e^{l_i - \log m},$$

where $m = \max_{j \in [n]} p_j$.

Task b

Lets first define the requested functions:

```
p <- function(x) exp(-x^2/2)/sqrt(2*pi)
lp <- function(x) -x^2/2-0.5*log(2*pi) # log(p)
l_P <- function(l,a=rep(1,length(l))) sum(a*l)
l_S <- function(l) { m <- max(l) ; m+log(sum(exp(l-m))) }
```

Without any tricks NaN is produced:

```
p(100)/(p(100)+p(100.01))
```

```
## [1] NaN
```

With the log trick we get the correct answer:

```
exp(l_P(c(lp(100),-l_S(c(lp(100),lp(100.01))))))
```

```
## [1] 0.7310684
```

Alternatively, we can define new binary "logarithmic" +, -, and * operators if we do not like functions:

```
`%L*%` <- function(x,y) x+y
`%L/%` <- function(x,y) x-y
`%L+%` <- function(x,y) { m <- max(x,y); m+log(sum(exp(x-m)+exp(y-m))) }

exp(lp(100) %L/% (lp(100) %L+% lp(100.01)))
```

```
## [1] 0.7310684
```

Task c

In R large and small numbers typically evaluate to infinity (Inf) or zero.

```
print(exp(1000))
```

```
## [1] Inf
```

```
print(exp(-1000))
```

```
## [1] 0
```

`Inf` is infinity and `1/Inf` evaluates to zero.

```
1/Inf
```

```
## [1] 0
```

The smallest positive `x` such that `1+x!=1` is *machine epsilon* with the value of 2.220446×10^{-16}.

```
.Machine$double.eps
```

```
## [1] 2.220446e-16
```

```
1+.Machine$double.eps!=1
```

```
## [1] TRUE
```

```
1+.Machine$double.eps/2!=1
```

```
## [1] FALSE
```

Grading: It is not necessary to "show" that you have actually read the requested man pages, it is sufficient to know the answers to the above questions.

Grading

Points: max. 10 (a+b: 6, c: 4)

Problem 4

Task a

L_i are mutually independent a random variable with standard deviation of σ. The mean of n_{va} mutually independent random variables is (as shown in the lecture) unbiased estimate of the mean of L_i with the standard deviation of $\sigma/\sqrt{n_{va}}$, which is the requested result. (Variance would be standard deviation squared or σ^2/n_{va}.)

Grading: The above is sufficient answer, but I will provide "full" proof below for those interested.

L_i is a random variable with variance of σ^2. Define a new random variable $\Delta_i = L_i - E[L_i]$. This random variable has zero expectation $E[\Delta_i] = E[L_i] - E[L_i] = 0$ and variance of $E[\Delta_i \Delta_j] = \delta_{ij}\sigma^2$ (the variance does not change if the random variable is shifted by a constant amount, and the cross-variance is zero because of the data points in the validation set are mutually independent!). Here we have used the Kronecker delta, which obeys $\delta_{ii} = 1$ and $\delta_{ij} = 0$ if $i \neq j$.

Now we can write:

$$L^1 - E[L^1] = \sum\nolimits_{j \in S_{va}} (L^1_j - E[L^1_j])/n_{va} = \sum\nolimits_{j \in S_{va}} \Delta_i / n_{va}$$

The variance is then given by $\mathrm{Var}(L^1) = \sigma^2/n_{va}$, which can be derived as follows:

$$\begin{aligned}\mathrm{Var}(L^1) = E[(L^1 - E[L^1])^2] = E\left[(\sum\nolimits_{j \in S_{va}} \Delta_i/n_{va})^2\right] = E\left[\sum\nolimits_{j \in S_{va}} \sum\nolimits_{j' \in S_{va}} \Delta_j \Delta_{j'}/n_{va}^2\right] \\ = \sum\nolimits_{j \in S_{va}} \sum\nolimits_{j' \in S_{va}} E[\Delta_j \Delta_{j'}]/n_{va}^2 = \sigma^2/n_{va}^2 \sum\nolimits_{j \in S_{va}} \sum\nolimits_{j' \in S_{va}} \delta_{jj'} = \sigma^2/n_{va}.\end{aligned}$$

Task b

The "true loss" for (actually any) function f_κ is given by $L = E_{(x,y)\sim F}[L(y, f_\kappa(x))]$, which is the expected loss on new datapoint sampled from F. We can approximate L by the test set loss given by $\hat{L} = \sum_{i \in S_{te}} L(y_i, f_\kappa(x_i))/n_{te}$. The estimator is unbiased if $E[\hat{L}] = L$.

The data points in test set have been chosen in random from data points sampled i.i.d. from the distribution F, from which it follows that the data points in the test set are from distribution F as well. We can therefore write:

$$E[L(y_i, f_\kappa(x_i))] = E_{(x,y)\sim F}[L(y, f_\kappa(x))] = L,$$

from which it follows that

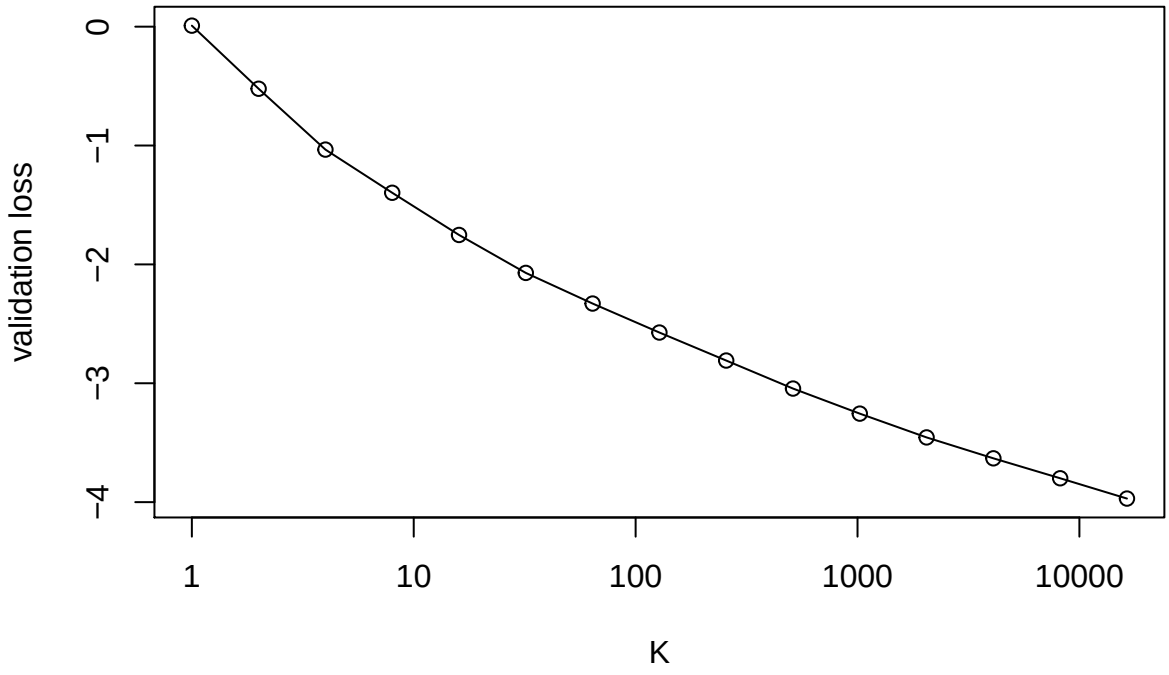
$$E[\hat{L}] = \sum_{i \in S_{te}} E[L(y_i, f_\kappa(x_i))]/n_{te} = \sum_{i \in S_{te}} L/n_{te} = L,$$

i.e., the estimator is unbiased.

Task c

The idea here is to study numerically the bias of the validation loss, if we want to estimate the true loss. Assume that we have K models to choose from which all have the "true loss" of zero and that the validation error has a variance of 1 (compare to task a above). Furthermore, assume that the validation errors for the K models are mutually independent and obey normal distribution. We can then plot the bias of the estimator as a function of K:

```
f <- function(K,iter=1000) mean(replicate(iter,min(rnorm(K))))
K <- 2^(0:14)
fK <- sapply(K,f)
plot(K,fK,xlab="K",ylab="validation loss",log="x")
lines(K,fK)
```



If there is only one model ($K = 1$) the estimator is unbiased, but more models there is to choose from, more the loss of the best model tends to underestimate the true loss. This is why it is important to use the test set to estimate the true loss of our supervised learning method, instead of using the validation loss (to estimate the loss on yet unseen data)!

Task d

Grading: This was a bonus task with no points awarded and therefore no need to grade.

Grading

Points: max. 15 (a: 5, b: 5, c: 5, d: 0)

Problem 5

Lets first define some auxiliary functions:

```
## the underlying function f(x)
f <- function(x) 1+x-x^2/2 # the underlying function

## make a dataset of n items
md <- function(n) {
  a <- data.frame(x=runif(n,min=-3,max=3))
  a$y <- f(a$x)+0.4*rnorm(n)
  a
}

## partition integers from 1 to n in random to k roughly equally sized parts.
kpart <- function(n,k,random=TRUE) {
  p <- if(random) sample.int(n) else 1:n
  s <- floor(seq(from=0,to=n,length.out=k+1))
  mapply(function(i,j) p[i:j],s[1:k]+1,s[2:(k+1)],SIMPLIFY=FALSE)
}

## perform k-fold cross validation
cv <- function(data,class,form=formula(sprintf("%s ~ .",class)),
               model=lm,k=10,split=kpart(nrow(data),k)) {
  ## define auxiliary function f that trains a model on subset itr items
  ## and outputs predictions for a subset iva items
  f <- function(itr,iva) {
    predict(model(form,data[itr,,drop=FALSE]),
            newdata=data[iva,,drop=FALSE])
  }
  yhat <- data[,class]
  ## go through all folds and estimate y on a model created by data form
  ## all other folds
  for(iva in split) {
    yhat[iva] <- f(setdiff(1:nrow(data),iva),iva)
  }

  ## output the CV estimate of y created by the above process
  yhat
}
```

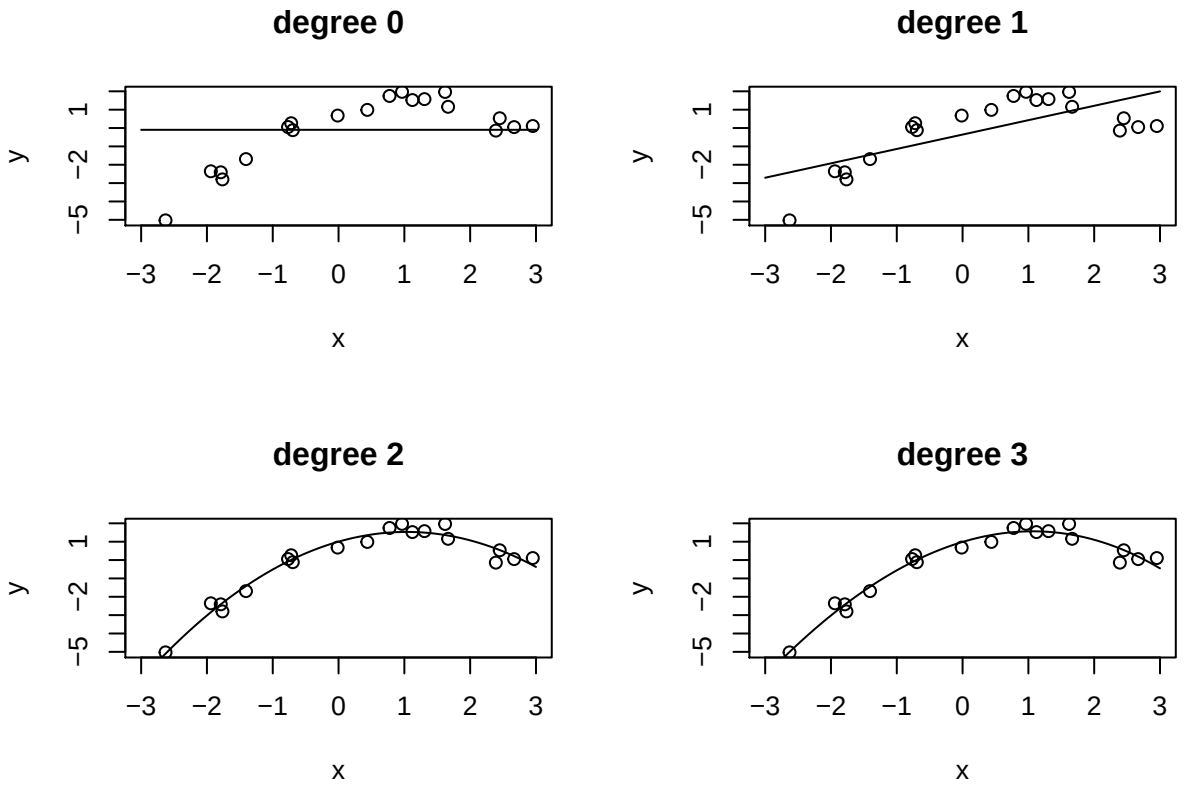
Task a

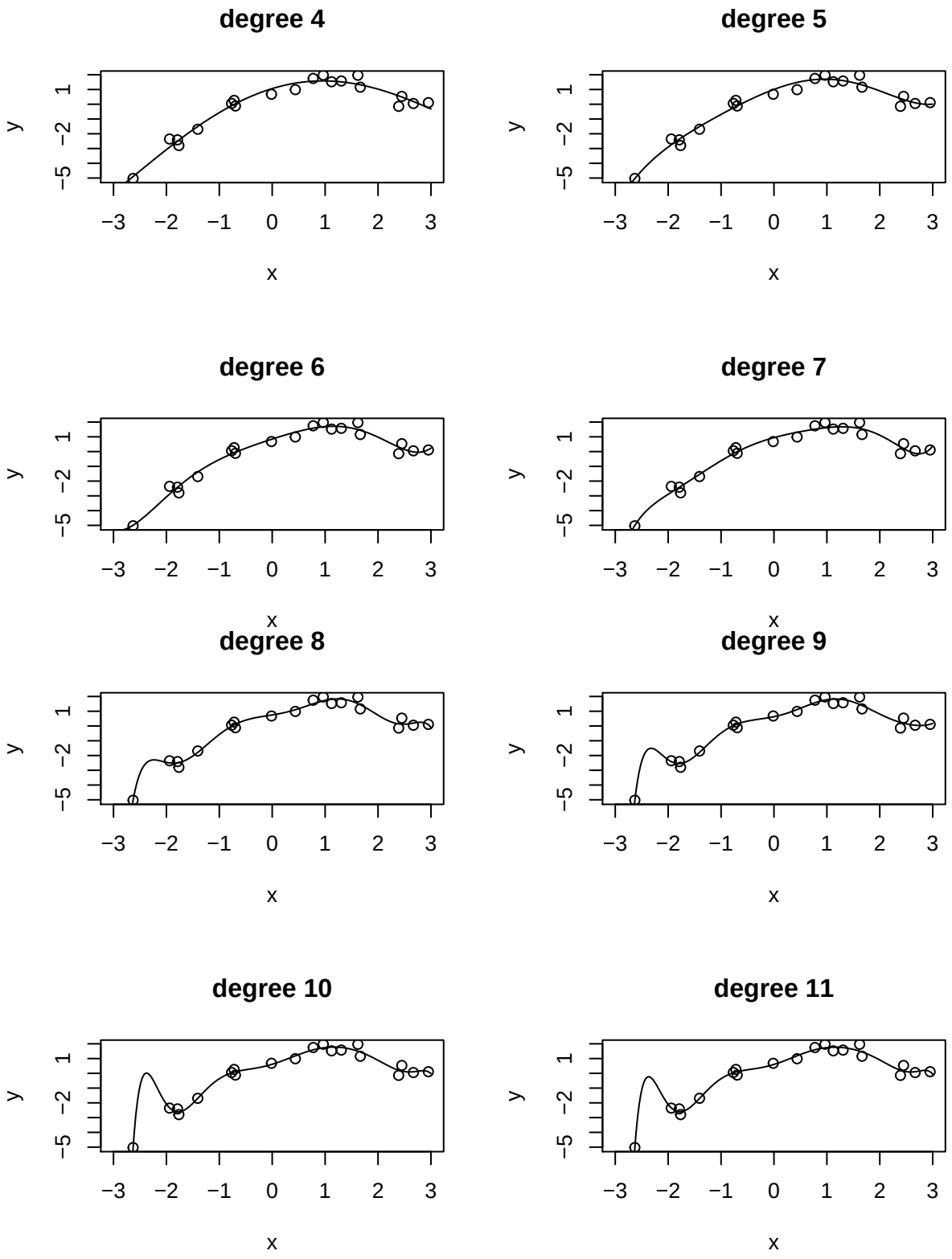
Generate the training, validation, and test dataset as instructed, first setting random seed so that we get the same results every time.

```
set.seed(1)
```

```
data <- md(1040)
Str <- 1:20
Sva <- 21:40
Ste <- 41:1040
```

```
makeplot <- function(degree,...) {
  x <- seq(from=-3,to=3,length.out=200)
  m <- lm(if(degree==0) formula(y ~ 1) else formula(y ~ poly(x,degree)),
          data[Str,])
  plot(c(-3,3),range(data[Str,"y"]),type="n",xlab="x",ylab="y",...)
  points(data[Str,])
  lines(x,predict(m,data.frame(x=x)))
}

for(degree in 0:11) {
  if(degree %% 4 == 0) par(mfrow=c(2,2))
  makeplot(degree,main=sprintf("degree %d",degree))
}
```





We simplify our task by first making a table where rows corresponds to polynomial degrees from 0 to 10 and the columns the following quantities which cover everything asked in the tasks: TRTR = training set loss for a model trained on the training set, VATR = validation set loss for a model trained on training set, TETR = test set loss for a model trained on training set, TETRVA = test set loss for a model trained on training+validation set, and CV = cross validation loss (on training+validation set).

```r
## MSE
mse <- function(y,yhat) mean((y-yhat)^2)

## make row to our data table
makerow <- function(degree) {
  form <- if(degree==0) formula(y ~ 1) else formula(y ~ poly(x,degree))
  ## train model on training set
  m.tr   <- lm(form,data[Str,])
  ## train model on training+validation set
  m.trva <- lm(form,data[c(Str,Sva),])

  c(TRTR=mse(data[Str,"y"],predict(m.tr,data[Str,])),
    VATR=mse(data[Sva,"y"],predict(m.tr,data[Sva,])),
    TETR=mse(data[Ste,"y"],predict(m.tr,data[Ste,])),
    TETRVA=mse(data[Ste,"y"],predict(m.trva,data[Ste,])),
    CV=mse(data[c(Str,Sva),"y"],cv(data[c(Str,Sva),],"y",form)))
}

res <- data.frame(degree=0:10,t(sapply(0:10,makerow)))

knitr::kable(res,"simple")
```

degree	TRTR	VATR	TETR	TETRVA	CV
0	3.2964966	4.6728121	5.0448883	4.9118552	4.1203628
1	1.5762837	1.9078886	2.1654342	2.0649218	1.8741480
2	0.1145753	0.1696184	0.1743727	0.1741473	0.1620879
3	0.1130214	0.1829990	0.1767914	0.1758663	0.1556514
4	0.1097449	0.1953651	0.1845318	0.1760929	0.1927286
5	0.0936311	0.1703806	0.1907097	0.1870235	0.1662638
6	0.0829667	0.2273480	0.2081371	0.1877443	0.1866211
7	0.0782424	0.2304756	0.2421751	0.1917056	0.2531418
8	0.0538810	5.9287818	3.6237922	0.2029514	0.2767821
9	0.0498533	18.9297571	11.0465059	0.2034269	0.2794018
10	0.0476071	74.1142695	42.4604000	0.2017701	0.2592090

Task b

The losses on the traing set are reported in column TRTR. We notice that these losses are always smaller for larger polynomial degrees.

Task c

The validation and test losses are reported in columns VATR and TETR, respectively. The validation and test set losses are both smallest for degree 2 polynomials, which is consistent with the fact that data is created by a degree 2 polynomial plus noise.

Task d

The degree $\kappa = 2$ polynomial has the smallest loss on the validation set. We would expect that E[TRTR]<E[VATR]<E[TETR]. Due to variance in estimators this expectation does not however always hold (recall that, e.g., validation set has only 20 items so there is some variance).

When we train a new regressor on training+validation set the error of the test set (column TETRVA) decreases sligthly to 0.1741473 from 0.1741473 for the regressor trained on the training set only (column TETR), as

expected.

Task e

The 10-fold CV losses are shown in column CV. The loss is smallest for the degree 3 polynomial, which is the degree we should choose if we would trust the CV result. The corresponding test set loss for a model trained on training+validation set is 0.1758663 (column TETRVA).

We also notice that the CV loss does not "blow up" as fast as the validation loss when the degree grows large, which is because CV is able to use the data more effectively.

The moral of the story is that the losses behave roughly as expected, but because of finite sample size and variance in the estimator the relative ordering of the various losses varies a bit, at least if the losses are close enough. And that cross-validation makes it possible to use the training+validation data in a more economical manner.

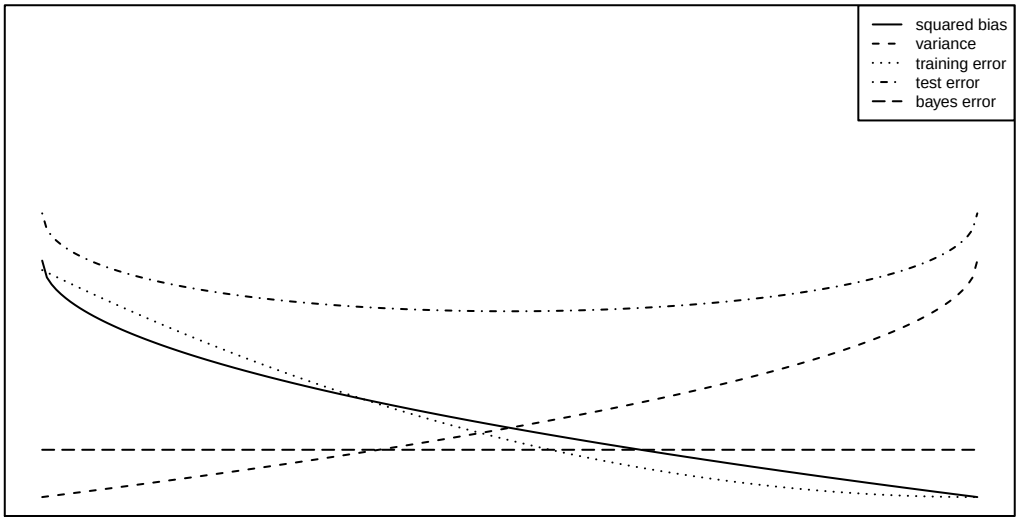
Grading

Points: max. 20 (a: 4, b: 4, c: 4, d: 4, e: 4)

The answer should get full points if the asked for numbers from the table have been produced, as well as figures requested, and there additionally an answer to direct question asked (i.e., in e, degree of polynomial to choose).

Problem 6

Task a



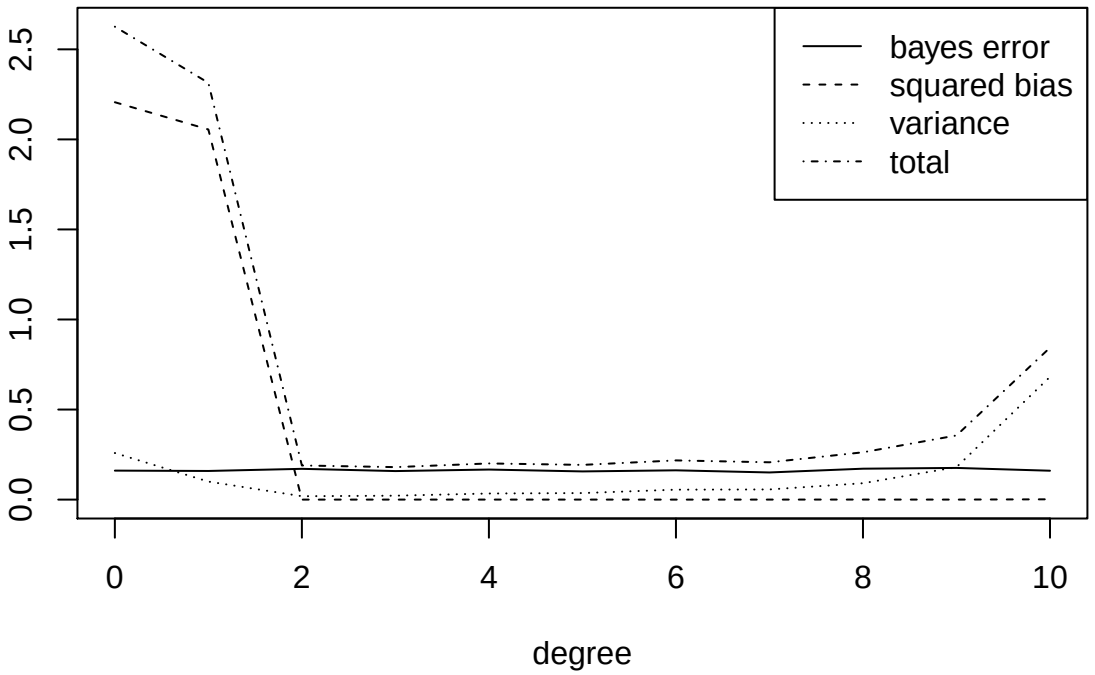
In the figure the models with low flexibility are on the left and large flexibility on the right. The Bayes error should be constant, because it does not depend on the model. The Bayes error gives a lower bound for a generalization error of any model. The squared bias is large for inflexible models and small for flexible models an the opposite is true for variance. The test error for regressors with MSE loss is sum of the Bayes error, squared bias, and variance and it is large for small flexibility (under-fitting) and large flexibility (over-fitting), having the minimum in between. The training error is usually always smaller than the error in the test set and it tends to decrease as model flexibility grows and go close to zero for very flexible models.

Task b

Here we study the bias variance decomposition at $x = 0$.

```
set.seed(1)


bv <- function(degree,iter=1000) {
  ## Here we create a dataset of 20 points, train a regression function, and predict
  ## the value at $x=0$; we do this 1000 times. For each of the 1000
  ## datasets we get a prediction g(0) from the regression function.
  g <- replicate(iter,
                     predict(lm(if(degree>0) formula(y~poly(x,degree)) else formula(y~1),
                                md(20)),

                             data.frame(x=0)))
  ## The 1000 test data points at (0,y0)
  y0 <- f(0)+0.4*rnorm(iter)

  ## Bayes error is here sigma^2 or about 0.4.
  sigma2 <- mean((y0-f(0))^2)
  ## (squared) bias (E[g]-f(0))^2
  bias2 <- (mean(g)-f(0))^2
  ## variance E((g-E[g])^2)
  variance <- mean((g-mean(g))^2)
  ## total = sigma2+bias2+variance
  total <- sigma2+bias2+variance
  ## total0 = E[(y0-g)^2]
  total0 <- mean((y0-g)^2)
  c(sigma2=sigma2,bias2=bias2,variance=variance,total=total,total0=total0)
}

BVD <- data.frame(degree=0:10,t(sapply(0:10,bv)))
```

```
plot(c(0,10),c(0,max(BVD$total)),type="n",xlab="degree",ylab="")
lines(BVD[,c("degree","sigma2")],lty="solid")
lines(BVD[,c("degree","bias2")],lty="dashed")
lines(BVD[,c("degree","variance")],lty="dotted")
lines(BVD[,c("degree","total")],lty="dotdash")
legend("topright",c("bayes error","squared bias","variance","total"),
       lty=c("solid","dashed","dotted","dotdash"))
```




```
knitr::kable(BVD,"simple")
```

degree	sigma2	bias2	variance	total	total0
0	0.1608857	2.2063268	0.2585569	2.6257694	2.6778249
1	0.1589162	2.0551699	0.0999985	2.3140845	2.3724097
2	0.1708348	0.0000059	0.0186821	0.1895227	0.1950247
3	0.1584206	0.0000313	0.0217756	0.1802276	0.1862036
4	0.1668471	0.0000377	0.0338662	0.2007510	0.1945398
5	0.1564011	0.0000339	0.0362595	0.1926945	0.1965123
6	0.1625204	0.0000000	0.0551275	0.2176479	0.2037655
7	0.1507474	0.0000661	0.0563517	0.2071652	0.1978199
8	0.1715630	0.0003062	0.0907959	0.2626651	0.2601906
9	0.1758911	0.0001397	0.1801694	0.3562002	0.3392019
10	0.1602385	0.0015205	0.6816070	0.8433661	0.8237424

In the table above, the rows correspond to polynomial degrees and the columns to the squared Bayes error (bayes2), squared bias (bias2), variance (variance), and their sum (total) as well as the MSE (total0). All of the values have been computed by sampling 1000 datasets, fitting a polynomial into them, and observing how the prediction $g(0)$ and $x = 0$ changes compared to the test data point at $(0, y)$.

We can first verify Eq. (2.7) of James et al. by observing that the columns total and total0 are roughly equal.

The bias-variance decomposition behaves as expected. Bayes error is roughly constant and squared bias decreases and variance increases as flexibility (here polynomial degree) increases.

Grading

Points: max. 20 (a: 10, b: 10)

Problem 7

t-statistic is the estimate (here of the mean) $\hat{\mu}$ divided by its standard deviation. Recall that the standard deviation of the estimate of the mean $\hat{\mu} = \sum_{i=1}^{n} y_i/n$ is $\mathrm{sd}(\hat{\mu}) = \sigma/\sqrt{n}$, where σ is the standard deviation of y_i.

```
sdhatmu <- sd(data[Str,"y"])/sqrt(length(Str))
tstat <- mean(data[Str,"y"])/sdhatmu
tstat95 <- c(tstat-1.96,tstat+1.96)
y95 <- tstat95*sdhatmu
```

The t-statistic is therefore -0.2314456. The 95% confidence interval is given by $t \pm 1.96 \times \mathrm{sd}(\hat{\mu})$ which corresponds to interval [-0.9128096,0.7200001]. Because zero is within the interval we cannot reject the null hypothesis that the mean is zero (with p=0.05).

Notice that we could also do the "real t test" in which above would be automatised:

```
t.test(data[Str,"y"])
```

```
##
##  One Sample t-test
##
## data:  data[Str, "y"]
## t = -0.23145, df = 19, p-value = 0.8194
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  -0.9682186  0.7754090
## sample estimates:
##   mean of x
## -0.09640477
```

We get the same value for the t-statistic. The 95% confidence interval, which in this case contains zero.

NB: Notice that the 95% confidence interval given by `t.test` is slightly wider that what we obtained above. This is because a mean of normally distributed variables obeys Student's t-distribution which has heavier tails than the normal distribution and for which the 95% confidence interval for 19 degrees of freedom (the degrees of freedom being here the size of data set minus one) is given by ± 2.093 standard deviations (not ± 1.96 standard deviations, as for the normal distribution). The ± 1.96 standard deviations however gives the correct 95% confidence intervals also for the Student't t-distribution at the limit of infinitely large data (at the limit $n \to \infty$ the Student's t distribution becomes normal distribution). ± 1.96 vs. ± 2.093 has however little practical significance here, because both of the numbers are "approximately 2" and the underlying assumptions (such as normality of random variables overaged over) are in practice only approximately satisfied anyway.

Grading: In this problem it is sufficient to use "approximately 2" to compute the 95% confidence interval from the t-statistic.

Task b

```
##
## Call:
## lm(formula = y ~ 1, data = data[Str, ])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.9321 -0.4317  0.2818  1.3447  2.0679
##
## Coefficients:
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -0.0964     0.4165  -0.231    0.819
##
## Residual standard error: 1.863 on 19 degrees of freedom
```

```
##
## Call:
## lm(formula = y ~ poly(x, 2, raw = TRUE), data = data[Str, ])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.76478 -0.20392 -0.00174  0.24986  0.60202
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)               1.00217    0.12453   8.048 3.37e-07 ***
## poly(x, 2, raw = TRUE)1  1.02274    0.05168  19.791 3.55e-13 ***
## poly(x, 2, raw = TRUE)2 -0.49397    0.03354 -14.727 4.14e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3671 on 17 degrees of freedom
## Multiple R-squared:  0.9652, Adjusted R-squared:  0.9612
## F-statistic: 236.1 on 2 and 17 DF,  p-value: 3.97e-13
```

```
##
## Call:
## lm(formula = y ~ poly(x, 3, raw = TRUE), data = data[Str, ])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.77033 -0.22484 -0.01727  0.24264  0.56024
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)               0.98767    0.13118   7.529 1.21e-06 ***
## poly(x, 3, raw = TRUE)1  1.07249    0.11854   9.047 1.09e-07 ***
## poly(x, 3, raw = TRUE)2 -0.48634    0.03800 -12.799 8.04e-10 ***
## poly(x, 3, raw = TRUE)3 -0.01043    0.02224  -0.469    0.645
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3759 on 16 degrees of freedom
## Multiple R-squared:  0.9657, Adjusted R-squared:  0.9593
## F-statistic: 150.2 on 3 and 16 DF,  p-value: 6.276e-12
```

```
##
## Call:
## lm(formula = y ~ poly(x, 5, raw = TRUE), data = data[Str, ])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.52759 -0.15857 -0.01406  0.23103  0.61045
##
## Coefficients:
```

```
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)               1.0051254  0.1696285   5.925 3.70e-05 ***
## poly(x, 5, raw = TRUE)1  1.3351511  0.1970626   6.775 8.95e-06 ***
## poly(x, 5, raw = TRUE)2 -0.5023993  0.1256455  -3.999  0.00132 **
## poly(x, 5, raw = TRUE)3 -0.1575955  0.0939306  -1.678  0.11556
## poly(x, 5, raw = TRUE)4  0.0003247  0.0169190   0.019  0.98496
## poly(x, 5, raw = TRUE)5  0.0154164  0.0099318   1.552  0.14292
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3657 on 14 degrees of freedom
## Multiple R-squared:  0.9716, Adjusted R-squared:  0.9615
## F-statistic: 95.78 on 5 and 14 DF,  p-value: 2.557e-10
```

The results indicate that we can reject the null hypothesis that the polynomial coefficients up to the 2nd degree are zero, but the null hypothesis that the higher degree coefficients are non-zero cannot be rejected, which is consistent with the fact that the data comes from degree 2 polynomial plus noise. The estimates for coefficients are also roughly correct, but start getting worse for higher degree polynomials. The "true" values for the polynomial coefficients are: $w_0 = w_1 = 1$, $w_2 = -1/2$, and $w_p = 0$ for $p \geq 3$.

Grading

Points: max. 15 (a: 7, b: 8)

Problem 8

Grading

Points: max. 5

Give full points if the answer contains some sentences that are about the topic of the question.