Machine Learning: Assignment 1

Karsten Standnes - STNKAR012

October 2017

1 Introduction

In this report I look at over-fitting and how noise,the amount of data and the complexity influence over-/under- fitting of data generated by an nth degree polynomial. The polynomials used are Legendre polynomials which are expressed in equation 3 below. In "Task 1" an introduction to the Legendre polynomials are presented and their use in the target function used in this project. "Task 2" simulate a fitting problem by generating data sets from a Legendre polynomial and adding different amount of noise it. Then different ordered Legendre polynomials are fitted to the data set.

Contents

2 Task 1

The target function used in "Task 1" can be represented in the following ways:

$$f(x) = \sum_{q=0}^{Q_f} \alpha_q x^q \tag{1}$$

or

$$f(x) = \sum_{q=0}^{Q_f} \beta_q L_q(x) \tag{2}$$

,$L_q(x)$ being the Legendre polynomial of q-th order. The Legendre polynomial is given by:

$$L(x) = 2^q \sum_{k=0}^{q} x^k \binom{q}{k} \binom{\frac{q+k-1}{2}}{q} \tag{3}$$

. The Legendre polynomials has the property that they are orthogonal for $x \in [-1, 1]$ (having weighting function 1, ref. β_q in equation 2).

2.1 Task 1 i

Plotting under the Legendre polynomial of the 0th to 5th order:

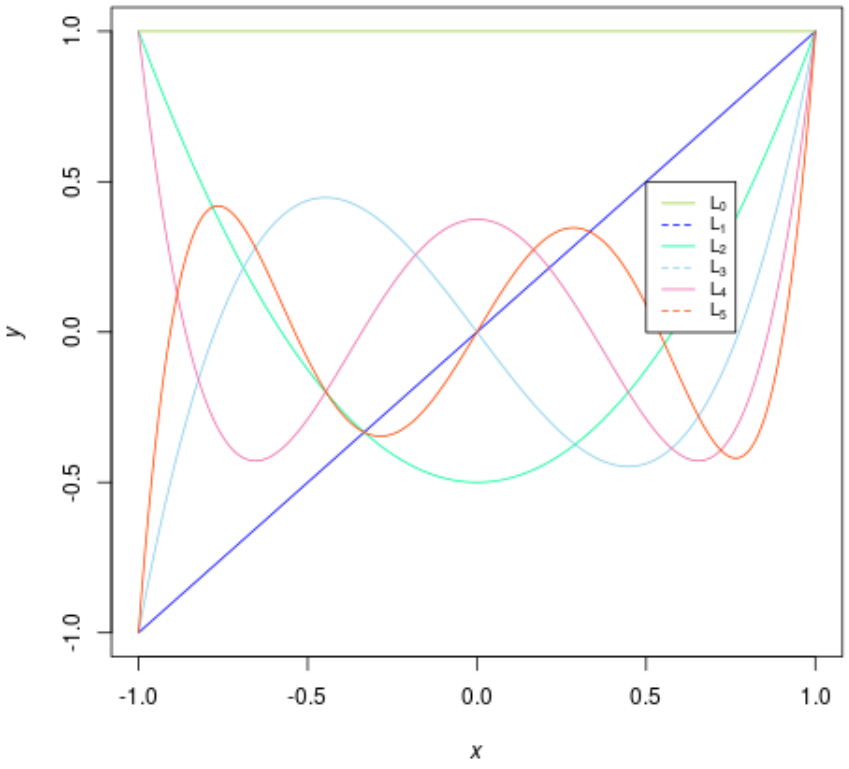
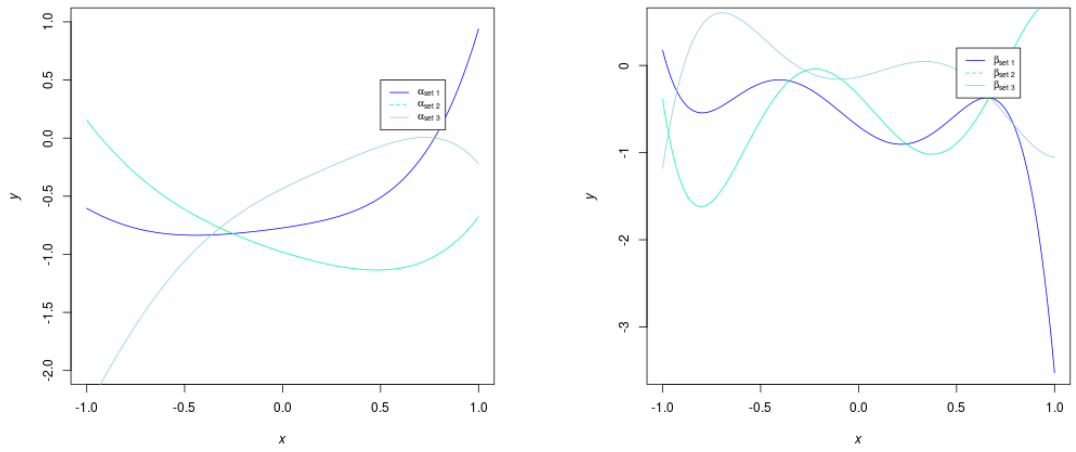


Figure 1: Legendre polynomials with order $q \in [0, 5]$

We can see that the Legendre polynomials are continuously defined on the interval $x \in [-1, 1]$ with y-values ranging from -1 to 1. We can also notice that the polynomials have symmetric and anti-symmetric properties. The even number polynomials are symmetric about the y-axis and the odd polynomials are anti-symmetric about the y-axis(being symmetric if mirrored over the x-axis). Also notice that the odd polynomials all intersect in the origin.

2.2 Task 1 ii

Below, two sets of polynomials are plotted, each of the sets consist of three 5th degree polynomials. The left ones are made by target function 1 with three randomly sampled α_q and the right ones equivalently created by target function 2 with randomly sampled sets of β_q.



(a) Target function from equation 1.

(b) Target function from equation 2.

Figure 2: Plots showing target functions.

From the figures above we can see a clear difference in shape of the polynomials made by the two target functions. The polynomials from function 1 are less oscillating the polynomials made by function 2 featuring Legendre polynomials. This can tell us something about the difference between the different types of polynomials. Even though they are of the same order, the behavior is quite different. And this can lead to it being hard to use a targe function to make a good fit to a data set created by a different target function.

3 Task 2

The target functions 1 and 2 can be used to give output y-values by using:

$$y = f(x) + \epsilon(x); \quad -1 \leq x \leq 1 \tag{4}$$

$$\text{where} : f(x) \text{ is a polynomial of order } Q_f$$
$$\text{and } \epsilon(x) \sim \text{Normal}(0, \sigma^2)$$

In this task this equation is used in combination with target function 2 to create data sets $\mathcal{D}$ containing sets of (x_i, y_i) with $i \in [i, N]$. x_i is randomly sampled from the uniform distribution from -1 to 1. The corresponding y_i is calculated by the target function of Q_f order and added a noise level σ. Then two sets of hypothesis are proposed to fit the data:

$$\mathcal{H}_2: \text{ 2nd-order polynomials}$$

$$\mathcal{H}_{10}: \text{ 10th-order polynomials}$$

In this task two random Legendre-based polynomial of 2nd degree $g_2^{\mathcal{D}}(x) \in \mathcal{H}_2$ and 10th degree $g_{10}^{\mathcal{D}}(x) \in \mathcal{H}_{10}$ are used to fit the data. Then a measure of over-fitting is used based on

$$\mathbb{E}_{\mathcal{D}}[E_{out}(g_{10}^{\mathcal{D}}) - E_{out}(g_2^{\mathcal{D}})] \tag{5}$$

where the out-of-sample error of each model is calculated and then the error from $g_2^{\mathcal{D}}(x)$ is subtracted from the error $g_{10}^{\mathcal{D}}(x)$. E_{out} is the same as the bias squared plus the noise.

3.1 Task 2 i

In this task a random 10th order target function $tf(x)$ is generated over $-1 \leq x \leq 1$ using the the Legendre based target function 2. This is done by creating a random set of β_q where $q \in [0, Q_f]$ with Q_f being the order of the target function. The β_q values are sampled from the uniform distribution from -1 to 1. Two vectors are created $\{20, 20 + \Delta N, ...120 - \Delta N, 120\}$ and $\{0.2, 0.2 + \Delta\sigma, ..., 1.1 - \Delta\sigma, 1.1\}$ where σ is the noise level and N is the number of data points. These are used to create different data sets from $tf(x)$. The combinations of σ and N is used to create a $K \times L$ matrix where K is the number of different σ's and L the number of N's. For each position in the matrix a 2nd and 10th order Legendre based polynomial $g_2^{\mathcal{D}}(x)$ and $g_{10}^{\mathcal{D}}(x)$ are calculated. This is done by predicting the β coefficients for the polynomials. The value for the position in the matrix is calculated by using equation 5. To avoid an anomaly in the results 40 runs are done with a new set of β's each time and then the values are averaged.

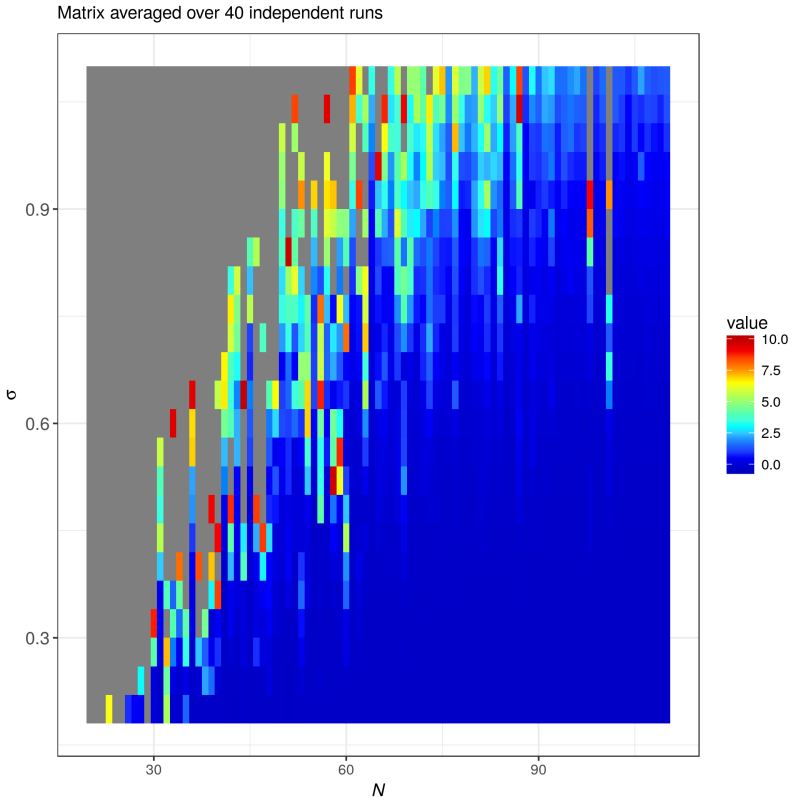


Figure 3: $\sigma \times N$ matrix with overfit measure plotted. Grey data is outside the color limit.

In the plot above it is a clear pattern that has emerged after 40 runs. There is a triangular pattern with the measure of overfit going towards 0 as the number of data points N increase and the variance σ is small. The grey values are values exceeding the threshold of 10 on the color bar. These high values are generated when the $g_{10}^{\mathcal{D}}(x)$ overfit the data set making the out of sample error $E_{out}(g_{10}^{\mathcal{D}})$ very large. Around $N = 60$ around half of the values in the that column gives "unstable" (high) values. From the plot it is also possible to see that a low N or a high σ is a good indication of a high value for the relative performance of the overfitting. A variance of 0.9 needs around 70 or more data points to give a "stable" (low) value, while a N of 35-40 need a variance around 0.25 or lower. On the other side of the spectre a set with $\max(N) = 110$ gives a "stable" performance even with $\max(\sigma) = 1.1$.

3.2 Task 2 ii

Instead of looking on σ and N, we can see how the order of the polynomial in the original target function affect the overfitting. There are more factors than σ and N that affect if the model fits well or not. When the dimensionality of the function a given data set is created from exceed the dimensionality of the function that is tried fitted, it will struggle. This is because it can't perfectly fit the function. This is just like a straight line may not perfectly fit a 2nd order polynomial, but on a relative small interval it can in some cases give a good estimate. Like in the previous task a matrix is made, but instead of using σ, combinations of the order of $tf(x)$ (Q_f) and N is used to fill the matrix. Then a number of runs is performed (here: 20) and as before the average of the runs is taken and one matrix is left. σ is set to 0.2 for all iterations.

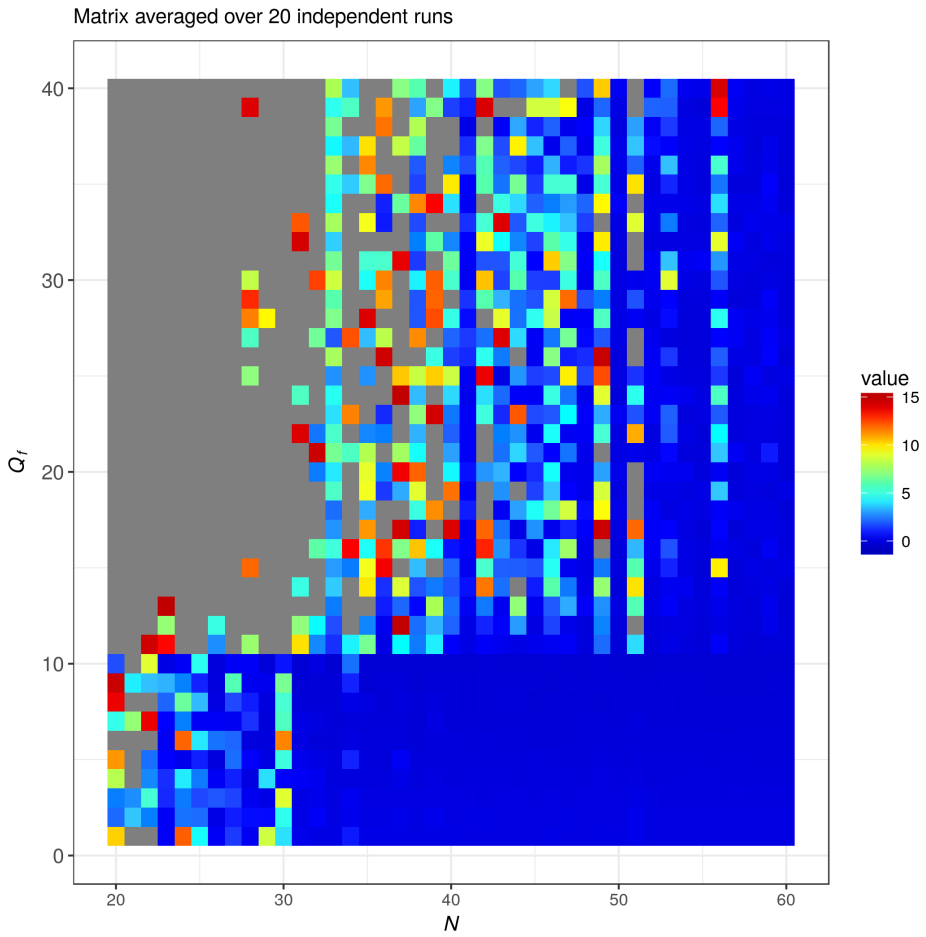


Figure 4: $Q_f \times N$ matrix with overfit measure plotted. Grey data indicate the values is outside the color limit.

From the plot above we can see some interesting trends in the pattern. As in "Task 2 i" we still use $g_2^{\mathcal{P}}(x)$ and $g_{10}^{\mathcal{P}}(x)$ to fit the data set. The highest order polynomial in the functions fitted is therefor 10. We can see that this is the case in the plot where there is a clear line or shift in the measured overfit between $Q_f = 10$ and $Q_f = 11$. This implies that the model more easily overfit when the original target function $tf(x)$ exceeds the functions tried fitted in dimensionality. It's also observed that with $50 < N \leq 60$ the overfit decreases even for large Q_f. This indicates that even though the high order($30 < Q_f$) of $tf(x)$ where the function oscillates a lot, the polynomials still performs fairly well with a high number of data points and low variance.

3.3 Task 2 iii conclusion

As seen above several factors affect the risk for overfitting a data set. As seen in figure 3 the chance of overfitting is directly linked to the number of data points and the size of the variance or noise level. This makes sense since fitting a polynomial to a data set with very few points makes it easier for the polynomial to fit to a completely different polynomial than the one we try to find. The lesser the points, the harder it is to see the pattern of the the underlying function. From the same figure it is also clearly visible that the variance has a great impact on the grade of overfitting. Since higher variance gives a higher noise level making all the points fitted to further from the underlying function. This makes it so that it seems like there is a more unclear pattern than there really is between the data and thereby increasing the chance of overfitting to this pattern. It's also worth noting that combining low amount of data points and high variance makes the overfit measure increase very

thereby indicating strong overfitting which makes sense together with the observations above.
From figure 4 another factor can be drawn to have impact on the grade of overfitting to the data, here referring to the grade Q_f. There is a clear border in plot when the complexity of the original target function exceeds the complexity of the functions attempted to fit the data. This meaning is very clear when the N is low and the fitting functions don't pick up that it's trying to fit to a higher complexity model and thereby overfits. The functions then try to make a model of it's own order and with few data points this easily produces a overfitted model. We see that when N increase the measure of overfit decrease and the fit is much better even for high orders of Q_f. This leads me to believe by combining the trends from both figure 3 and 4 that with a higher variance in "Task 2 ii" would give a higher chance of overfitting all over and vice versa.

4 Appendix

4.1 R-Code: Question 1

```
set.seed(1234)
## legendre expression within sum formula
l <- function(x,q,k){
      x**k * choose(q,k) * choose((q+k-1)/2,q)
    }

## produces legendre polynomial
legendres <- function(x,q)
    {
    func <- rep(0,length(x))
        for(k in 0:q){
            func <- func + l(x,q,k)
        }
        #print(func)
    func <- 2**q * func
    }

x = seq(-1,1, by = 0.01)

## plot legendre polynomial of 0 to 5th degree
plot1a <- function(){
    png("task1_plots/task1i.png")
    l0 <- legendres(x,0)
    l1 <- legendres(x,1)
    l2 <- legendres(x,2)
    l3 <- legendres(x,3)
    l4 <- legendres(x,4)
    l5 <- legendres(x,5)
    plot(x,l1, type = "l", col = "blue", ylab = expression(italic("y")),
        xlab = expression(italic("x")))
    lines(x,l2, col = "mediumspringgreen")
    lines(x,l3, col = "skyblue")
    lines(x,l4, col = "hotpink2")
    lines(x,l5, col = "orangered")
    lines(x,l0, col = "yellowgreen")
    legend(0.5,0.5, legend = c(expression("L"[0]),
                               expression("L"[1]),
                               expression("L"[2]),
                               expression("L"[3]),
                               expression("L"[4]),
                               expression("L"[5])),
        col = c("yellowgreen", "blue", "mediumspringgreen",
                "skyblue", "hotpink2", "orangered"), lty=1:2, cex=0.8)
    dev.off()
    }

## generate uniform values between -1 and 1
generateUniformValues <- function(n)
    {
      runif(n, min = -1, max =1)
    }

## target function 1
targetFunc1 <- function(x,Q_f)
    {
```

```
        alphas <- generateUniformValues(Q_f+1)
        func <- rep(0, length(x))
        for(q in 0:Q_f){
            func <- func + (alphas[q+1] * x**q)
        }
        func
    }

## target function 2
targetFunc2 <- function(x,Q_f)
    {
        betas <- generateUniformValues(Q_f+1)
        func <- rep(0,length(x))
        for(q in 0:Q_f){
            func <- func + betas[q+1] * legendres(x, q)
        }
        func
    }

## plot target function 1 and 2
plot3tf1tf2 <- function(){
      tf1_1 <- targetFunc1(x,5)
      tf1_2 <- targetFunc1(x,5)
      tf1_3 <- targetFunc1(x,5)

      tf2_1 <- targetFunc2(x,5)
      tf2_2 <- targetFunc2(x,5)
      tf2_3 <- targetFunc2(x,5)

      png("task1_plots/task1iialpha.png")
      plot(x,tf1_1, type = "l", col = "blue", ylim = c(-2,1),
           ylab = expression(italic("y")), xlab = expression(italic("x")))
      lines(x,tf1_2, col = "mediumspringgreen")
      lines(x,tf1_3, col = "skyblue")
      legend(0.5,0.5, legend = c(expression(alpha["set␣1"]),
                                 expression(alpha["set␣2"]),
                                 expression(alpha["set␣3"])),
             col = c("blue", "mediumspringgreen", "skyblue" ),
             lty=1:2, cex=0.8)
      dev.off()

      png("task1_plots/task1iibeta.png")
      plot(x,tf2_1, type = "l", col = "blue", ylim = c(-3.5,0.5),
      ylab = expression(italic("y")), xlab = expression(italic("x")))
      lines(x,tf2_2, col = "mediumspringgreen")
      lines(x,tf2_3, col = "skyblue")
      legend(0.5,0.2, legend = c(expression(beta["set␣1"]),
                                 expression(beta["set␣2"]),
                                 expression(beta["set␣3"])),
             col = c("blue", "mediumspringgreen", "skyblue"),
             lty=1:2, cex=0.8)
      dev.off()

    }
plot1a()
plot3tf1tf2()
```

4.2 R-Code: Question 2

```
library(plot3D)
library(ggplot2)
library(reshape2)     # for melt(...)
library(grid)         # for unit(...)
library(colorRamps)

set.seed(1234)

## inner expression of sum formula in Legendre polynom
l <- function(
    x,q,k
    ){
      x**k * choose(q,k) * choose((q+k-1)/2,q)
    }

## print legndre coefficients
l_coeff <- function(
    q,k
    )
    {
      print(choose(q,k) * choose((q+k-1)/2,q))
    }

## legendre polynomial
legendres <- function(
    x,q
    )
    {
    func <- rep(0,length(x))
        for(k in 0:q){
            func <- func + l(x,q,k)
        }
    func <- 2**q * func
    }

## function integrated over to find difference between orginal target
   function and the one fitted
difference <- function(
    x,
    betas10order,
    betas_g_fitted
    )
    {
        diff <- 0

        length_measure <- length(betas10order) - length(betas_g_fitted)
        if(length_measure > 0) {
            betas_g_fitted <- c(betas_g_fitted, rep(0, length_measure))
        }else{
          if(length_measure < 0){
            betas10order <- c(betas10order, rep(0,-length_measure))
          }
        }

        for(i in 1:length(betas10order)){
            diff <- diff + (betas10order[i] - betas_g_fitted[i])
                    * legendres(x,i)
```

```
        }
        diff^2
    }

## generate n values from uniform distribution with limits [-1,1]
generateUniformValues <- function(n)
    {
      runif(n, min = -1, max =1)
    }

## target function used, return corresponding y-values based on input x,
    order Q_f and betas
targetFunc2 <- function(
    x,
    Q_f,
    betas
    )
    {
        #betas <- generateUniformValues(Q_f+1)
        func <- rep(0,length(x))
        for(q in 0:Q_f){
            func <- func + betas[q+1] * legendres(x, q)
        }
        func
    }

## produce legendre polynomial values for the legendre polynomials from the
    target function
makeModelg <- function(
    order,
    x
    )
    {
        require(polynom)
        betas <- numeric(order)
        value_matrix <- matrix(0L, nrow = length(x), ncol = order+1)
        fitted_values <- matrix(0L, nrow = length(x), ncol = order+1)

        for(i in 0:order){
            value_matrix[,i+1] <- legendres(x, i)
        }

        value_matrix[,2:ncol(value_matrix)]
    }

## measure the overfit between a 10th- and 2nd order legendre polynomial
measureOverfit <- function(
    x_values,
    y_values,
    betasQorder,
    sigma,
    order1 = 10,
    order2 = 2
    )
    {
      y_2g <- lm(y_values~ makeModelg(2,x_values))
      y_10g <- lm(y_values ~ makeModelg(10,x_values))

      betasg2 <- unname(coef(y_2g))
```

```
    betasg10 <- unname(coef(y_10g))
    betasg2[is.na(betasg2)] <- 0
    betasg10[is.na(betasg10)] <- 0

    #increasing rel.tol to avoid round off error for high integrals
    biasg2 <- integrate(difference,-1,1,betasQorder,betasg2,
                        rel.tol = 1.5e-2)
    biasg2 <- biasg2$value

    biasg10 <- integrate(difference,-1,1,betasQorder,betasg10,
                         rel.tol = 1.5e-2)
    biasg10 <- biasg10$value

    err_g10 <- biasg10 + sigma^2
    err_g2 <- biasg2 + sigma^2

    E <- err_g10 - err_g2
    E
  }

## update the average values of the matrix with the values from the last run
update_matrix_avg <- function(
  current_avg,
  new_matrix,
  i
  )
  {
    matrix_list <- list(current_avg*(i-1), new_matrix)
    Reduce("+", matrix_list) / i
  }

## create error matrix for task 2 i
makeErrorMatrix_i <- function(
  N,
  sigma,
  betas10order
  ){
    m <- matrix(0L, nrow = length(N), ncol = length(sigma))
    for(k in 1:length(N)){
      x_values <- generateUniformValues(N[k])
      x_values <- sort(x_values)
      y_values <- targetFunc2(x_values, 10, betas10order)
        for(l in 1:length(sigma)){
            sigma_l <- sigma[l]
            y_values <- y_values + rnorm(n = length(x_values), mean = 0,
                        sd = sigma_l^2)
            m[k,l] <- measureOverfit(x_values, y_values, betas10order,
                      sigma_l)
        }
    }
    m
  }

## controll function for making matrices for task 2 i and averaging
avg_runs_i <- function(
  n_avgs,
  N,
  sigma
  )
```

```
    {
      m <- matrix(0L, nrow = length(N), ncol = length(sigma))
      for(i in 1:n_avgs){
          print(i)
          betas10order <- generateUniformValues(11)
          new_m <- makeErrorMatrix_i(N, sigma, betas10order)
          m <- update_matrix_avg(m, new_m, i)
      }
      print(min(m))
      print(max(m))
      print(m)
    }

############
# Task 2 i#
############
task2i <- function()
    {

      N = seq(20,110, by=1)
      sigma = seq(0.2, 1.1, by=0.04)

      m_error <- avg_runs_i(40, N, sigma) ## choose number of runs,
                                            N and sigma

      colnames(m_error) <- sigma
      rownames(m_error) <- N
      write.table(m_error,"N1sig004Run40.csv")

      plot_g10_minus_g2(m_error, N, sigma)

    }

## make error matrix for task 2 ii
makeErrorMatrix_ii <- function(
    N,
    sigma,
    Q_f
    ){
      m <- matrix(0L, nrow = length(N), ncol = length(Q_f))
      for(k in 1:length(N)){
        x_values <- generateUniformValues(N[k])
        x_values <- sort(x_values)
          for(l in 1:length(Q_f)){
              betasQorder <- generateUniformValues(Q_f[l]+1)
              y_values <- targetFunc2(x_values, Q_f[l], betasQorder) +
                          rnorm(n = length(x_values), mean = 0, sd = sigma
                              ^2)
              m[k,l] <- measureOverfit(x_values, y_values, betasQorder,
                                       sigma)
          }
      }
      m
    }

## controll function for making matrices for task 2 ii and averaging
avg_runs_ii <- function(
    n_avgs,
    N,
```

```
    Q_f,
    sigma
    )
    {
      m <- matrix(0L, nrow = length(N), ncol = length(Q_f))
      for(i in 1:n_avgs){
          print(i)
          new_m <- makeErrorMatrix_ii(N, sigma, Q_f)
          m <- update_matrix_avg(m, new_m, i)
          plot_g10_minus_g2_ii(m,1,1)
      }
      print(m)
    }

############
# Task 2 ii#
############
task2ii <- function()
    {
      N <- seq(20, 60, by=1)
      Q_f <- seq(1,40, by=1)
      sigma <- 0.2

      m_error <- avg_runs_ii(20, N, Q_f, sigma) ## choose number of runs, N,
                                                  Q_f and sigma

      colnames(m_error) <- Q_f
      rownames(m_error) <- N
      write.table(m_error,"N1Q_f1Run20.csv")
      print(max(m_error))
      print(min(m_error))

      plot_g10_minus_g2_ii(m_error,1,1)

    }

## plot overfit error matrix with the different sigmas and N's on the y and
   x axis
plot_g10_minus_g2 <- function(
    m_error,
    N,
    sigma
    )
    {
      gg <- melt(m_error)
      str(gg)

      ggplot(gg,aes(x=Var1,y=Var2))+
          #geom_tile(aes(fill = value)) +
          geom_raster(aes(fill = value)) +
          scale_fill_gradientn(colours = topo.colors(10),
                               limits = c(-0.5,10)) +
          labs(x="N", y=expression(sigma),
               title="Matrix averaged over 40 independent runs") +
          theme_bw() + theme(axis.text.x=element_text(size=9, angle=0,
                             vjust=0.3),
                     axis.text.y=element_text(size=11),
                     plot.title=element_text(size=11))
      ggsave("N1sig004Run40task2iSmooth.pdf")
```

```
      ggplot(gg,aes(x=Var1,y=Var2))+
          geom_tile(aes(fill = value)) +
          scale_fill_gradientn(colours = colorRamps::matlab.like2(10),
                               limits = c(-0.5,10)) +
          labs(x=expression(italic("N")), y=expression(sigma),
               title="Matrix averaged over 40 independent runs") +
          theme_bw() + theme(axis.text.x=element_text(size=9, angle=0,
                             vjust=0.3),
                     axis.text.y=element_text(size=11),
                     plot.title=element_text(size=11))
      ggsave("N1sig004Run40task2i.png")

      #pdf("3D-plot2.pdf")
      #      persp3D(x = N, y = sigma, z = m_error, colvar = m_error , clim
          = c(-10,10), zlim = c(-2,2))
      #dev.off()

    }
## plot matrix for task 2 ii
plot_g10_minus_g2_ii <- function(
    m_error,
    N,
    sigma
    )
    {
      gg <- melt(m_error)
      str(gg)

      ggplot(gg,aes(x=Var1,y=Var2))+
          #geom_tile(aes(fill = value)) +
          geom_raster(aes(fill = value)) +
          scale_fill_gradientn(colours = topo.colors(10),
                               limits = c(-0.5,3)) +
          labs(x=expression(italic("N")), y=expression(italic("Q"[f])),
                       title="Matrix averaged over 20 independent runs") +
          theme_bw() + theme(axis.text.x=element_text(size=9, angle=0,
                               vjust=0.3),
                     axis.text.y=element_text(size=11),
                     plot.title=element_text(size=11))
      ggsave("smoothedGridtaskii20.png")

      ggplot(gg,aes(x=Var1,y=Var2))+
          geom_tile(aes(fill = value)) +
          scale_fill_gradientn(colours = colorRamps::matlab.like2(10),
                               limits = c(-1,15)) +
          labs(x=expression(italic("N")), y=expression(italic("Q"[f])),
                       title="Matrix averaged over 20 independent runs") +
          theme_bw() + theme(axis.text.x=element_text(size=9, angle=0,
                               vjust=0.3),
                     axis.text.y=element_text(size=11),
                     plot.title=element_text(size=11))
      ggsave("normalGridtaskii20.png")

    }

## if matrix is stored in table format with named columns and rows, it can
    be loaded from disk and plotted
read_table_from_file <- function()
```

```
    {
      m_file <- as.matrix(read.table("N1sig01Run20.csv", check.names = FALSE
          ))
      print((m_file[1:20,1:20]))
      plot_g10_minus_g2(m_file, 1, 1)

      #m_file <- as.matrix(read.table("N1Q_f1Run10.csv", check.names = FALSE
          ))
      #print((m_file[1:20,1:20]))
      #plot_g10_minus_g2_ii(m_file, 1, 1)
    }

## (optional) plot target function with nois together with fitted models,
    good for viewing if the function fits are good or not
plot_run <- function(
    x_values,
    y_values,
    x_test_data,
    g10_y,
    g2_y
    )
    {
      plot(x_values, y_values,type = "b", col = "green")
      lines(x_test_data, g10_y, type = "b", col = "hotpink2")
      lines(x_test_data, g2_y, type="b", col = "yellowgreen")
      legend(0.8,0.5, legend = c("The orginal", "g10", "g2"), col = c("green
          ", "hotpink2", "yellowgreen"), lty = 1:2, cex=0.8)
    }


### RUN FUNCTIONS ###

task2i()
#read_table_from_file()
task2ii()
```