HW 01 - Statistical Learning

G21: Davide Mascolo - Antonella Cruoglio - Giuliana Iovino - Mario Napoli

29 marzo 2022

Part 1

1.

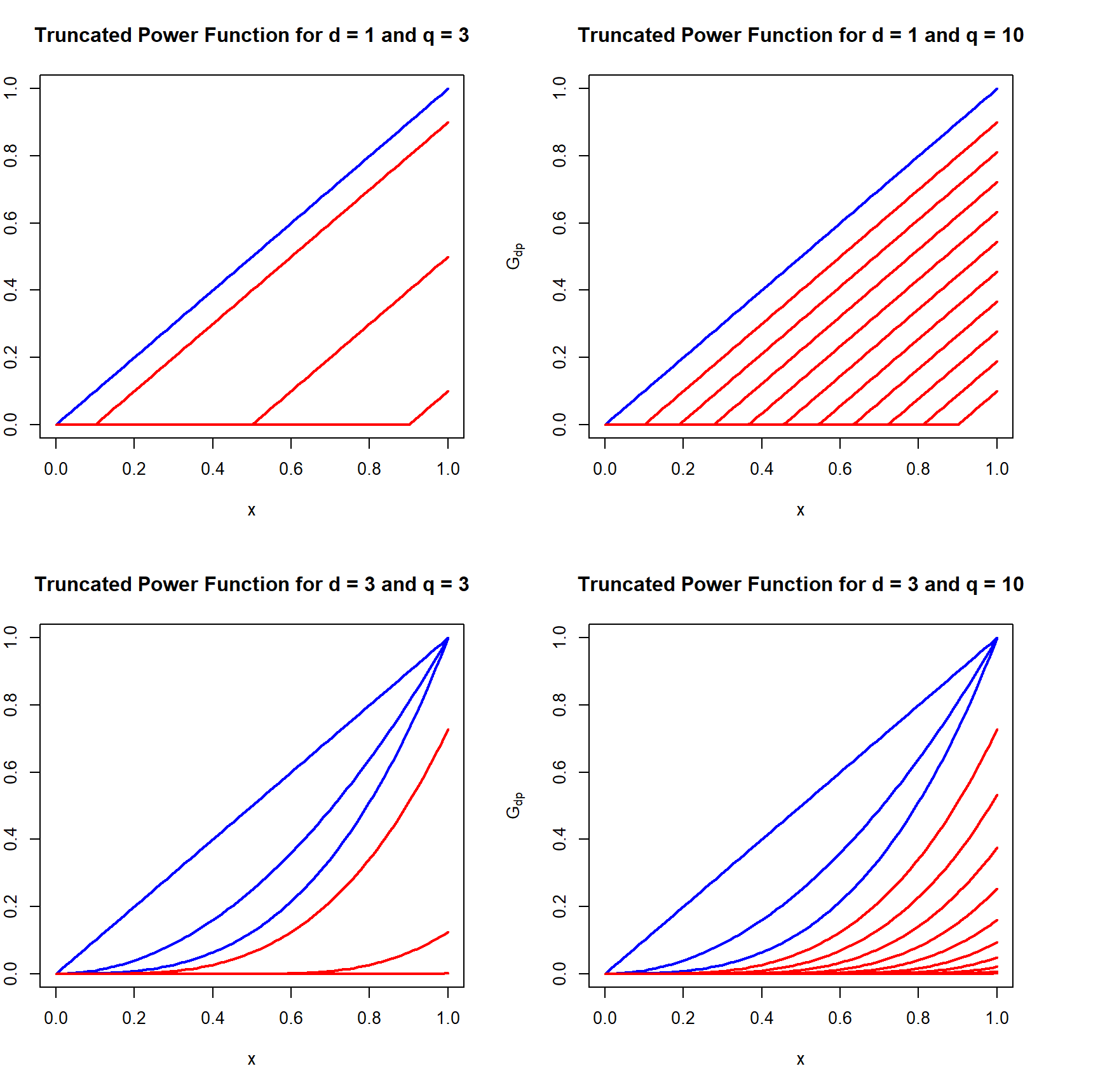
Unlike the orthogonal series expansion that model the global behavior of data, with the powered truncated functions we can model the local structures of data. More over truncated basis functions are neither normal neither orthogonal.

2.

We plot a few elements of $G_{d,q}$ with $d \in (1, 10)$ and $q \in (3, 10)$.

```
## Truncated Power Functions
g_blue <- function(x, d) x^(d)
g_red  <- function(x,eps,d) pmax(0, (x-eps))^d
```

```
## Plot
Gdq <- function(d,q){
  x <- seq(0,1,0.01)
  knots <- quantile(x, probs = seq(0.1, 0.9,
                                   length.out = q))

  plot.new()
  plot.window(xlim=c(0,1), ylim=c(0,1))
  axis(1)
  axis(2)
  title(paste('Truncated Power Function for d =', d,
            'and q =', q))
  title(xlab = 'x')
  title(ylab = parse(text = paste0('G[dp]')))
  box()

  for(d in 1:d) {
    g1 <- g_blue(x, d)
    curve(g_blue(x, d),
          add = TRUE, col = 'blue', lwd = 2)
  }

  knots <- quantile(x, probs = seq(0.1, 0.9,
                                   length.out = q))

  for(q in 1:q){

    g2 <- g_red(x,knots[q],d)
    curve(g_red(x,knots[q],d),
          add = TRUE, col = 'red', lwd = 2)
  }
}
```



3. Spline Regression

We implement regression splines from scratch by considering:

- $d \in (1, 3)$
- knots on q-equispaced locations within the x-range

```
## Design Matrix
design.mat <- function(data, d, q){
  n <- nrow(data)
  j <- (d+1)+q
  X <- matrix(nrow = n, ncol= j)
  X <- data.frame(X)

  for (i in 0:d) {
    X[,i+1] <- g_blue(data$x, i)
  }

  knots <- quantile(data$x, probs = seq(0.1, 0.9,
                                        length.out = q),
                    names = FALSE)


  for (k in 1:q) {
    X[,(d+1+k)] <- g_red(data$x, knots[k], d)
  }
  X$y <- data$y

  ## Esclude the intercept
  return(X[,-1])
}
```

3.1 Parameter tuning with K-Fold Cross Validation

We choose the best combination of d and q.

```
KFCV <- function(data, d, q, K){
  n <- nrow(data)

  folds <- sample(rep(1:K, length = n))
  KCV <- vector() ## Init the CV-score vector

  mseq <- rep(0, q)

  for (idx in 1:q) {
    train <- design.mat(data, d, idx)
    ## Loop
    for (k in 1:K){
      ## Fit using obs *not* in the active fold V_k
      fit    <- lm(y ~ ., data = train[folds != k,])
      ## Get the X's for obs in the active fold V_k
      x.out  <- train[folds == k, ]
      ## Predict on the obs in the active fold V_k
      yhat   <- suppressWarnings(predict(fit,
                                         newdata = x.out))
      ## Compare with the responses in the active fold V_k
      y.out  <- train$y[which(folds == k)]
      KCV[k] <- mean((y.out - yhat)^2)
    }
    mseq[idx] <- mean(KCV)    ## K-CV estimate

  }
  return(mseq)
}
```

```
## Optimal choice
optim <- function(d, K, mseq) data.frame(K = K, d = d,
                                  q = which.min(mseq),
                                  MSE = min(mseq))
```

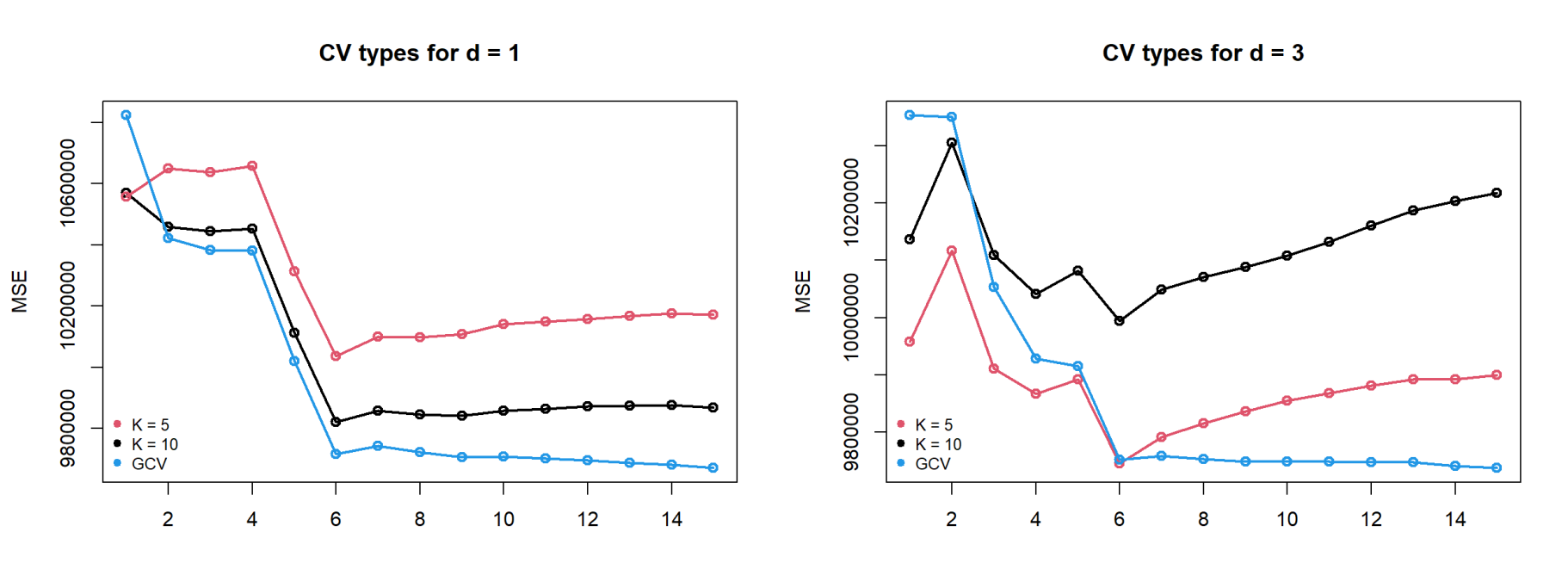
K	d	q	MSE
5	1	6	10037124
5	3	6	9745927
10	1	6	9821671
10	3	6	9993875

The best combination of parameters is with $d = 3$ and $q = 6$. These two values minimize the MSE and are obtained through 5Fold CV. We note that the value of MSE is relatively big, but this can depends by the range of the target variable. In this case we have a lot of variability in the data and we will see it later.

3.2 Parameter tuning with Generalized Cross Validation

```
## Implementation of GCV
Generalized_CV <- function(data, d, q){
  n <- nrow(data)
  p <- (d+1)+q

  GCVq <- rep(0, q)
  for (idx in 1:q) {
    train <- design.mat(data, d, idx)
    fit <- lm(y ~ ., data = train)
    MSE.tr <-deviance(fit)/n
    GCV <- MSE.tr/(1-(p/n))^2
    GCVq[idx] <- GCV
  }
  return(GCVq)
}
```

```
## Generalized Cross Validation
GCV_D1 <- Generalized_CV(data, 1, 15)
GCV_D3 <- Generalized_CV(data, 3, 15)
```



Looking at these plots we can compare the estimation error returned by the KFCV with the estimation error returned by the Generalized Cross Validation. In particular, we note how the error estimate using GCV tends to decrease as the number of nodes q increases. Evaluating this error would lead to the choice of a too high number of nodes and a model that would probably be overfitted. This is justified by the fact that the validation of the model using this technique takes place with a closed formula and therefore heavily underestimating the generalization error. For these reasons the optimal combination is the one obtained through 5Fold CV.

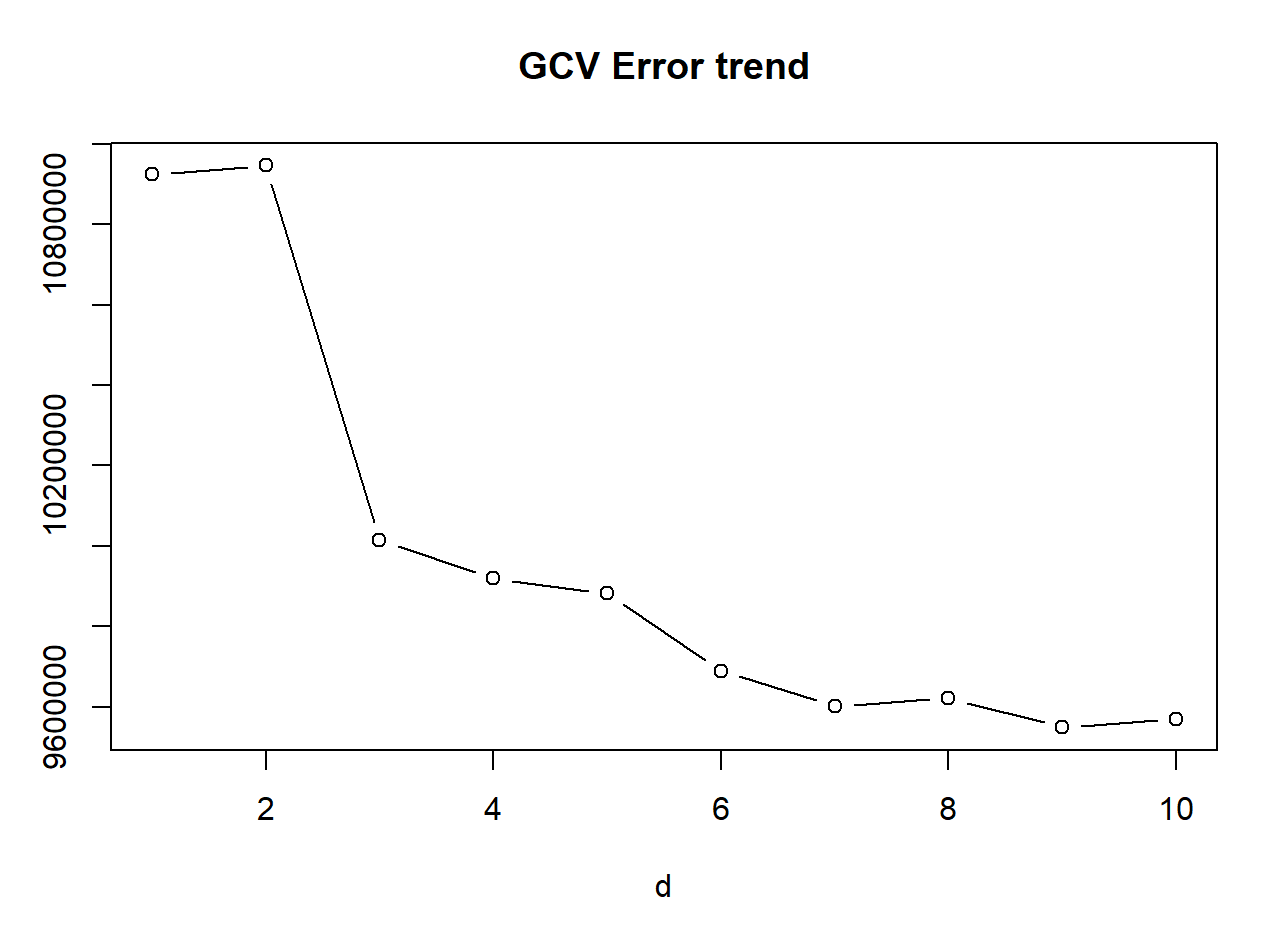
Now, we can use least squares to determine the optimal coefficients $\hat{\beta}$.

```
## Best combination: d = 3, q = 6
X    <- design.mat(data, d = 3, q = 6)
fit  <- lm(y ~ ., data = X)
(cf  <- fit$coefficients)
```

```
  (Intercept)            X2            X3            X4            X5
713.5918410114  4.5755096628  0.1963463854 -0.0004359675 -0.0005087852
           X6            X7            X8            X9           X10
 0.0023522450 -0.0023773504  0.0014745970 -0.0006177328  0.0031214911
```

GCV-tuned polynomial regression

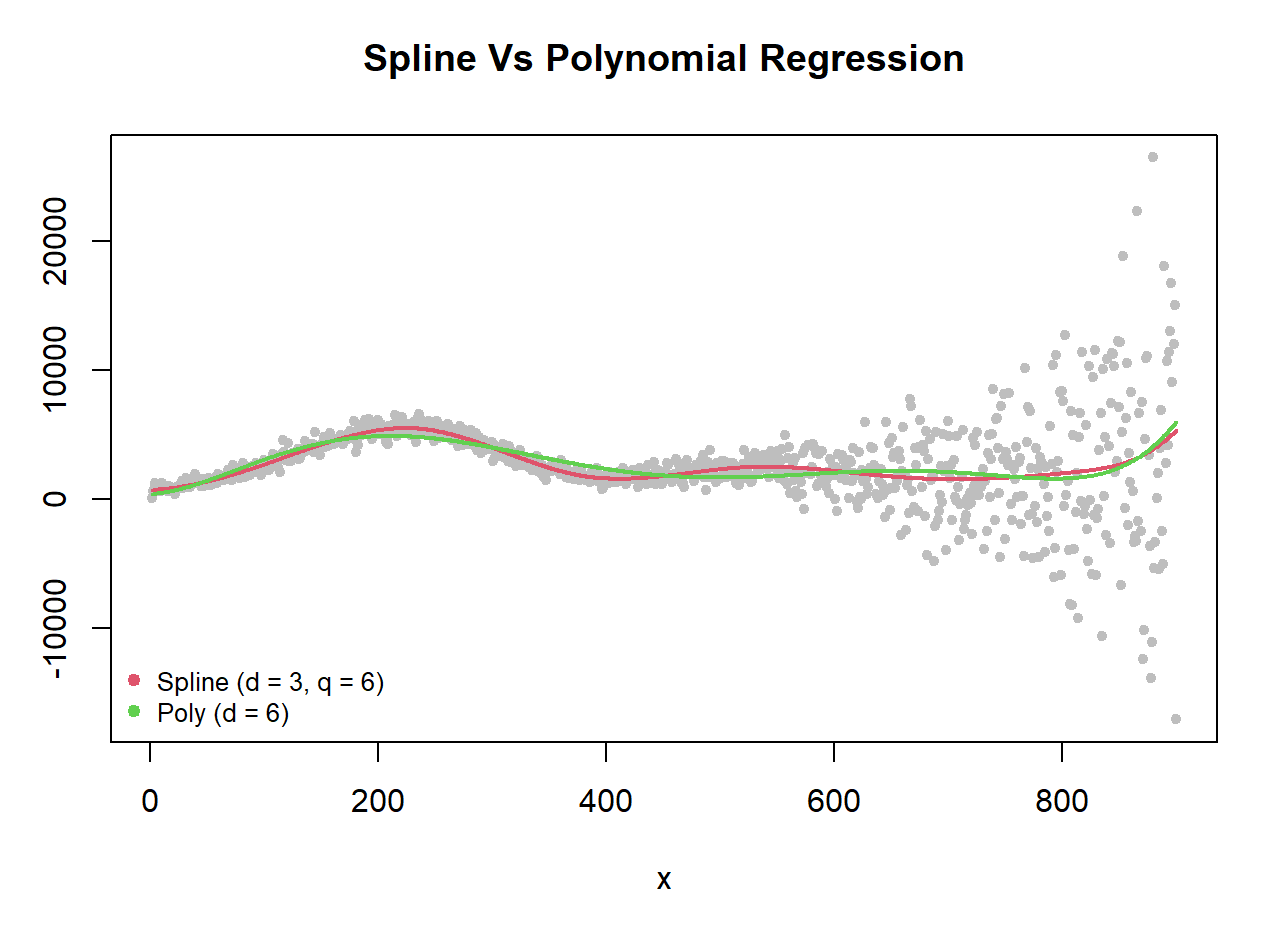
```
## Implementation of polynomial regression
Poly_GCV <- function(data, d, q){
    n <- nrow(data)
    p <- (d+1)+q

    GCVq <- rep(0, q)
    for (idx in 1:q) {
      fit    <- lm(y ~ poly(x, degree = d), data = train)
      MSE.tr <-deviance(fit)/n
      GCV    <- MSE.tr/(1-(p/n))^2
      GCVq[idx] <- GCV
    }
    return(GCVq)
  }
```



Considering what has been said above about the GCV problem, we decide to select $d = 6$ as the optimal value, since as d increases we would be led to always choose the last value considered.

4. Spline regression Vs. Polynomial regression

We compare the best spline fit with the GCV-tuned polynomial regression.



The distribution is strongly heteroskedastic, that is, with non-constant variance as the value of x increases. Graphically, heteroskedasticity can be seen from the cone structure of the distribution.

We can see that the two regressions have a similar general trend, although the spline regression is smoother and better approximates the distribution of the data at some points.

Also, spline regression can be modeled more by choosing the location of the nodes. In fact, given that the distribution is heteroskedastic and that the nodes have been selected equally spaced, it is possible to select a smaller number of nodes at the beginning of the distribution, since in those values the distribution is quite simple and then move more nodes towards the right side of the distribution, where the structure to be captured is more complicated.

Part 2

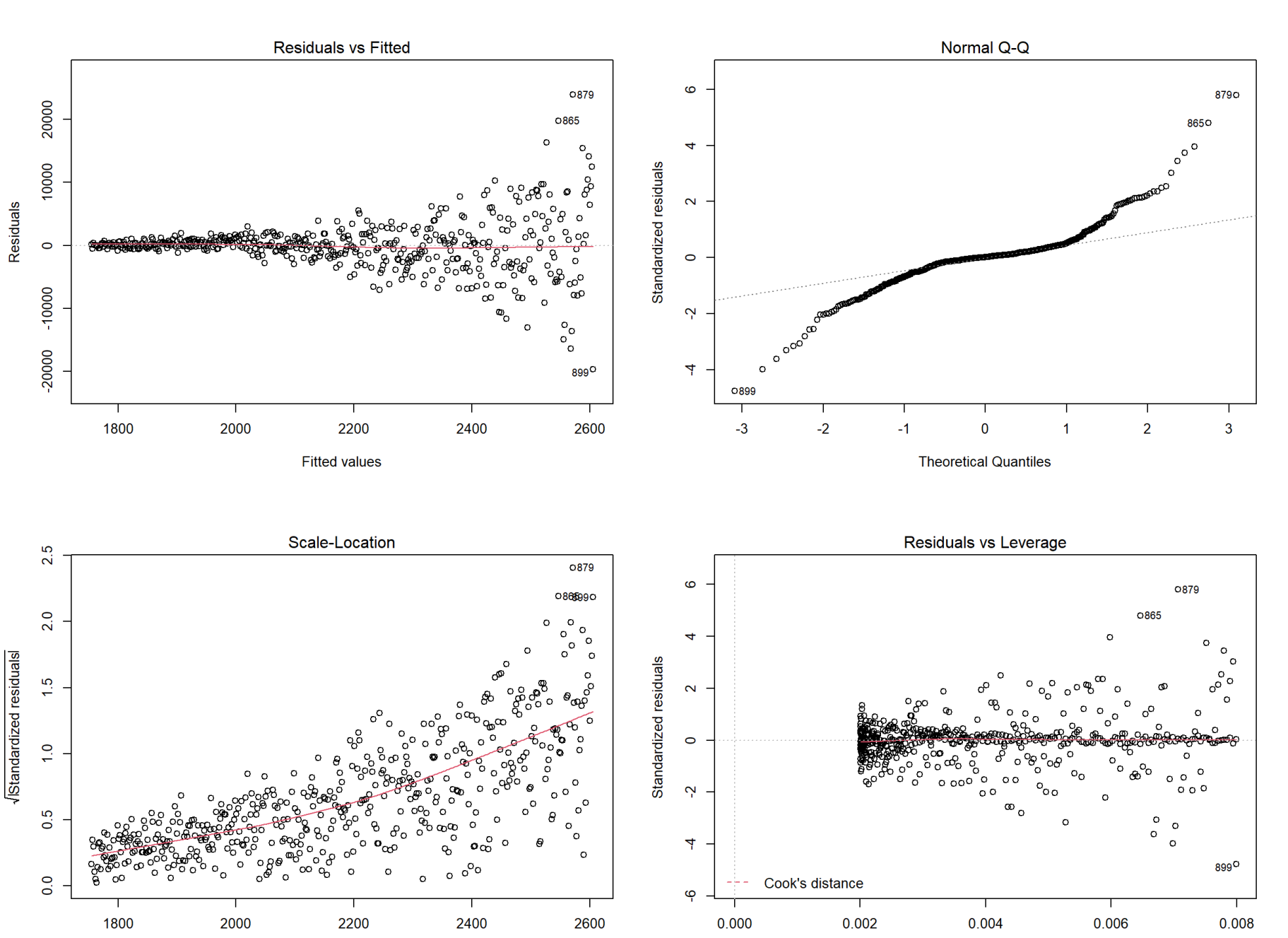
1.

We drop the first 400 observations and saving the others in a dataframe calles $wmap_{sb}$. Then, we consider a simple linear model $f(x|\theta) = \theta_0 + \theta_1\, x$.

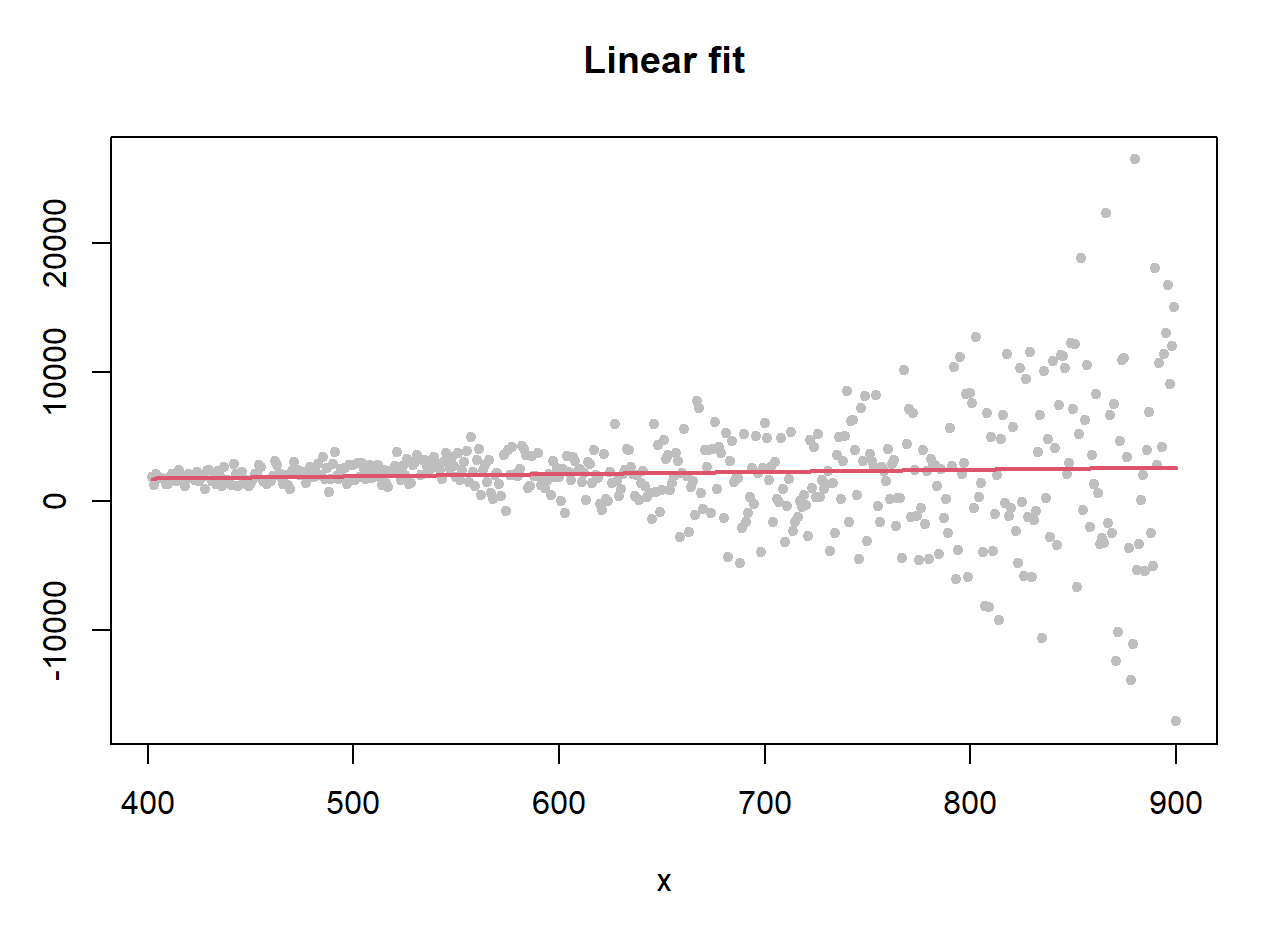
2. Fit simple linear model

```
Call:
lm(formula = y ~ x, data = wmap_sb)

Residuals:
     Min       1Q   Median       3Q      Max
-19651.3  -1311.8     60.4   1208.1  23909.8

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 1068.277    857.690   1.246    0.214
x              1.708      1.286   1.328    0.185

Residual standard error: 4139 on 497 degrees of freedom
Multiple R-squared:  0.003534,  Adjusted R-squared:  0.001529
F-statistic: 1.762 on 1 and 497 DF,  p-value: 0.1849
```

We can see that the hypothesized model is unable to capture the information contained in the data. In fact, the R^2 is close to zero, the range of residuals is very wide and the median is very different from zero, suggesting that there is no normal distribution of the residuals.



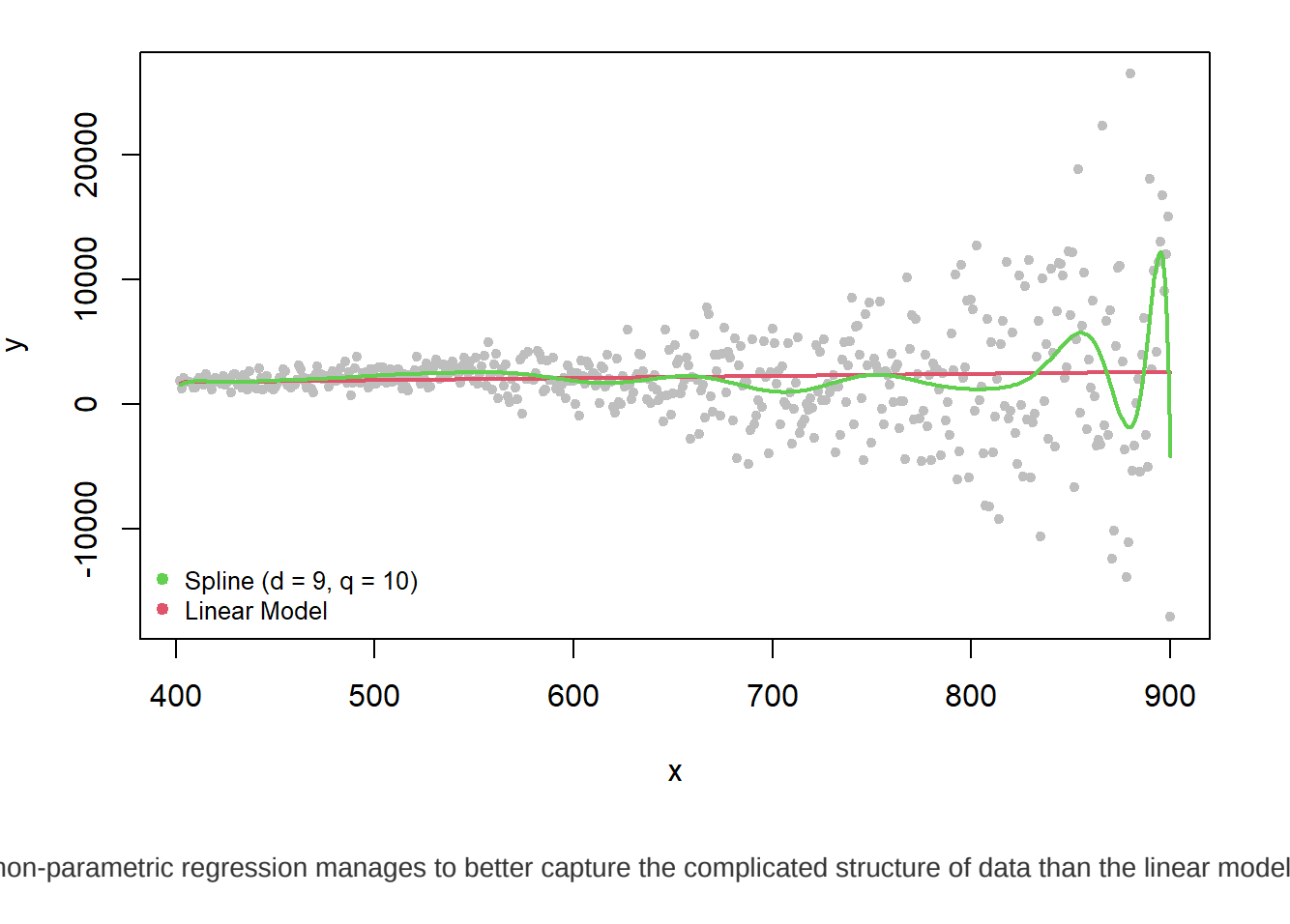
The QQ-Plot confirms the non-normal distribution of the residuals, while from the plot with the predicted values we can see their heteroskedastic structure.



We confirm what has been said, the model is too simple and it's unable to explain the target variable which has a lot of variability in the last observations.

3.

We add the non-parametric model to the plot that we made before.



It's evident that the non-parametric regression manages to better capture the complicated structure of data than the linear model.

4.

We calculate $\hat{t}$.

```
## Estimate t_hat
t_hat <- MSEp_hat - MSEnp_hat
cat(t_hat)
```

```
2178298
```

5.

Here we have a function to simulate new dataset.

```
## Inputs: linear model (lin_fit), x values at which to
## simulate(sim_x)
## Outputs: Data frame with columns x and y
sim_lm <- function(lin_fit, sim_x) {
  n <- length(sim_x)
  sim_fr <- data.frame(x = sim_x)
  sigma <- summary(lin_fit)$sigma
  y_sim <- predict(lin_fit, newdata = sim_fr)
  y_sim <- y_sim + rnorm(n, 0, sigma)     ## Add noise
  sim_fr <- data.frame(sim_fr, y = y_sim) ## Adds y column
  return(sim_fr)
}
```

```
## Bootstrap
B <- 1000
MSEp_tilde  <- rep(0, B)
MSEnp_tilde <- rep(0, B)
t_tilde     <- rep(0, B)

for (b in 1:B){
  ## New data
  X_new <- sim_lm(lin_fit, wmap_sb$x)
  ## Fit linear model
  lin_fit <- lm(X_new$y ~ ., data = X_new)
  ## MSE parametric model
  MSEp_tilde[b] <- mean((X_new$y -
                         lin_fit$fitted.values)^2)
  ## Generate design matrix
  X <- design.mat(wmap_sb, best_d, best_q)
  ## Fit the spline with degree = 3
  nlin_fit <- lm(X$y ~ ., data = X)
  ## MSE non parametric model
  MSEnp_tilde[b] <- mean((X_new$y -
                          nlin_fit$fitted.values)^2)
  ## Estimate T
  t_tilde[b] <- MSEp_tilde[b] - MSEnp_tilde[b]
}
```

6.

```
## Bootstrapped p-value
res <- mean(t_tilde > t_hat)
cat(res)
```

```
0
```

We reject the null hypothesis H_0 with a level of $\alpha = 0.05$, so we can say that the parametric model is not correct and this is coherent with what was said before. This means that there is some non-linear structure (bumps) into the distribution that cannot be captured by the linear model.