Week 5

gherardo varando

December 29, 2018

Some artificial experiments

Ex 1

1.1

We generate data from the model

$$X \sim N(0, \sigma_1^2)$$
$$Y|X = x \sim N(ax + b, \sigma_2^2)$$

```
sigma1 <- 2
sigma2 <- 3
a <- -1
b <- 6
n <- 50
```

Since the model is described on terms of the distribution of X and of the conditional distribution of Y given $X = x$ it is natural to sample first observations for X and then the corresponding observations of Y.

```
x <- rnorm(n, mean = 0, sd = sigma1)
```

Now we use the fact that $Y|X = x$ can be written as

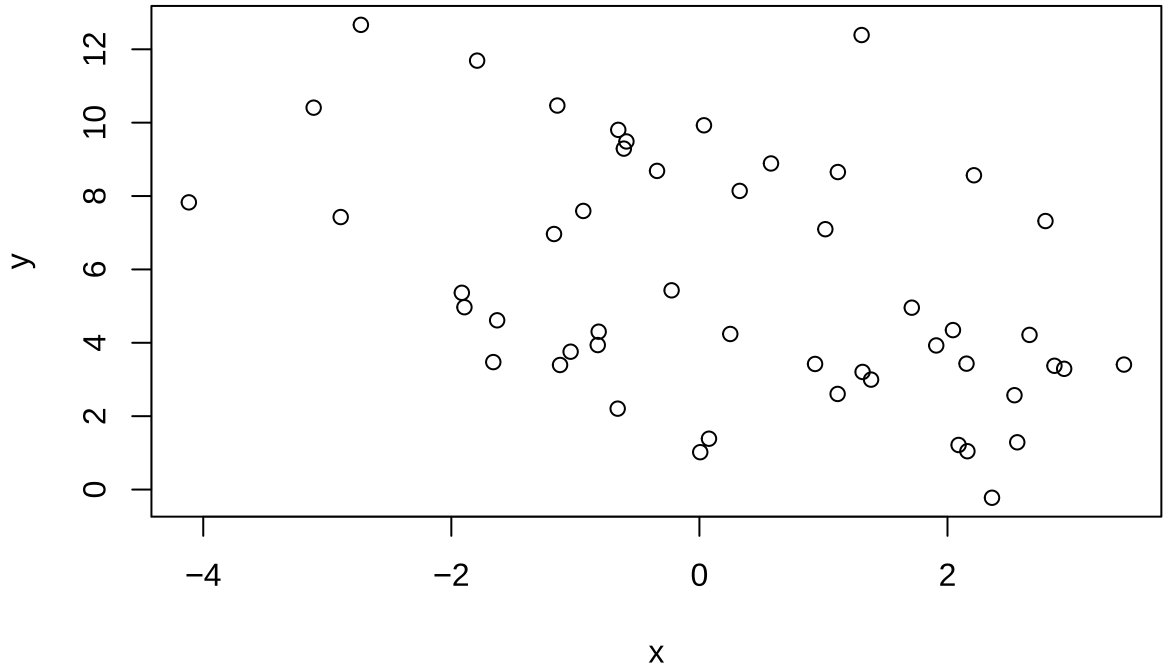
$$Y = ax + b + \sigma_2 Z$$

where $Z \sim N(0, 1)$ is a standard Gaussian random variable. This is a simple consequence of the property of the Gaussian distribution under scale-location transformations.

```
y <- a*x + b + sigma2 * rnorm(n)
### equivalently
## y <- a*x + b + rnorm(n, sd = sigma2)
### or also
## y <- a*x + rnorm(n, mean = b, sd = sigma2)
### or even the more expensive, but probably more intuitive
## y <- sapply(x, function(xx){
## return(rnorm(1, mean = a*x + b, sd = sigma2))
## })
```

we can now plot the observations in a scatter plot:

```
plot(x,y)
```

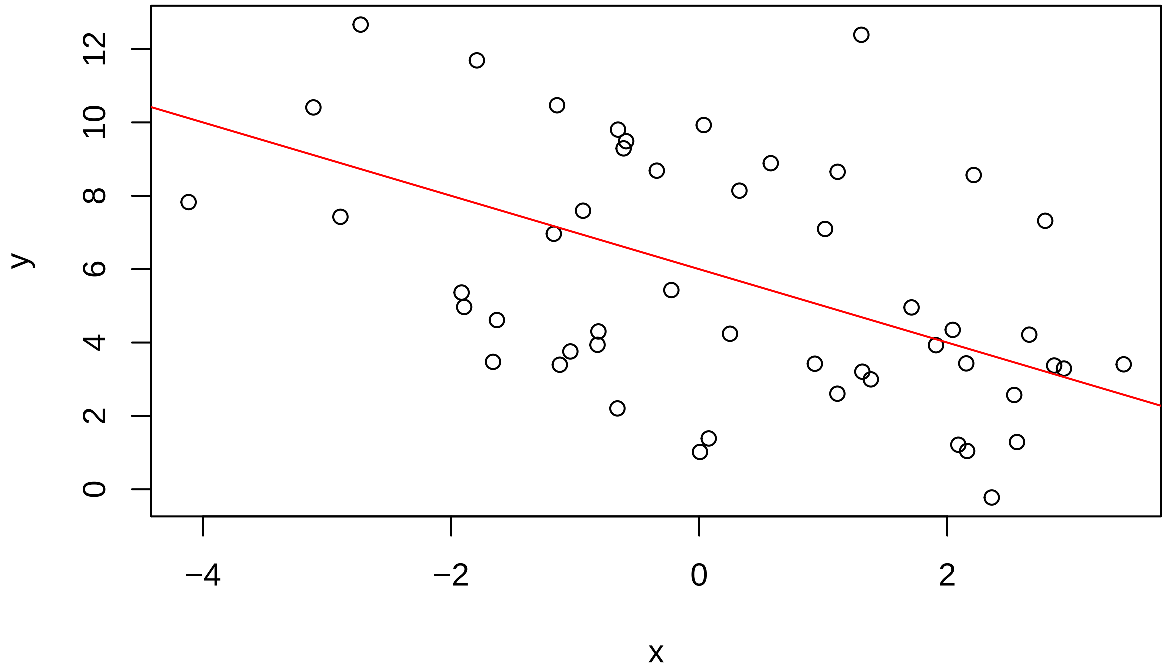


The true regression function is the function

$$r(x) = \mathbb{E}(Y|X = x) = \mathbb{E}(N(ax + b, \sigma_2^2)) = ax + b$$

We can use the `abline` function to plot it, be careful that we called the intercept with b and the slope with a

```
plot(x,y)
abline(a = b, b = a, col = "red")
```



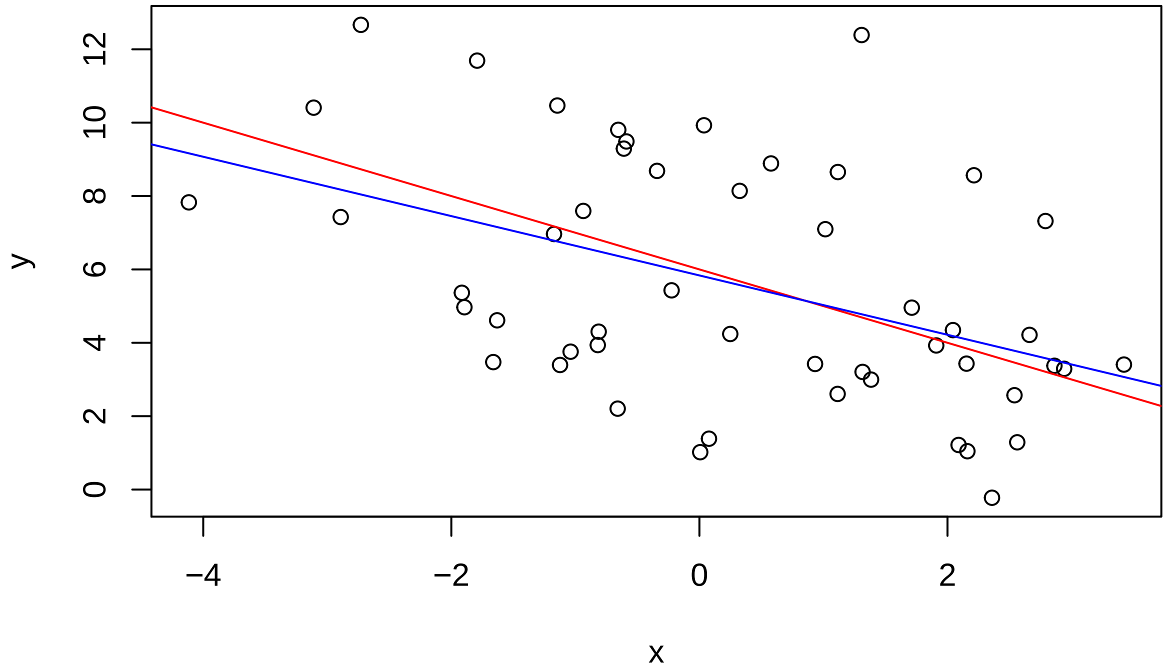
1.2

We fit now a linear regression model.

```
fit <- lm(formula = y ~ x, data = data.frame(x = x, y = y))
```

And we plot the fitted regression line on top of the scatter plot and alongside the true regression line.

```
plot(x,y)
abline(a = b, b =a, col = "red")
abline(fit, col = "blue")
```



1.3

We use the summary function to obtain information on the fitted coefficients.

```
summary(fit)
```

```
##
## Call:
## lm(formula = y ~ x, data = data.frame(x = x, y = y))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.8120 -2.3807 -0.4759  2.5717  7.6097
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   5.8360     0.4266  13.680  < 2e-16 ***
## x            -0.8087     0.2321  -3.484  0.00107 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.981 on 48 degrees of freedom
## Multiple R-squared:  0.2018, Adjusted R-squared:  0.1852
## F-statistic: 12.14 on 1 and 48 DF,  p-value: 0.001066
```

1.4

We repeat the data generation but now with a true intercept $b = 2$, since then we are asked to repeat the experiment with different parameter we will just write a function to generate data:

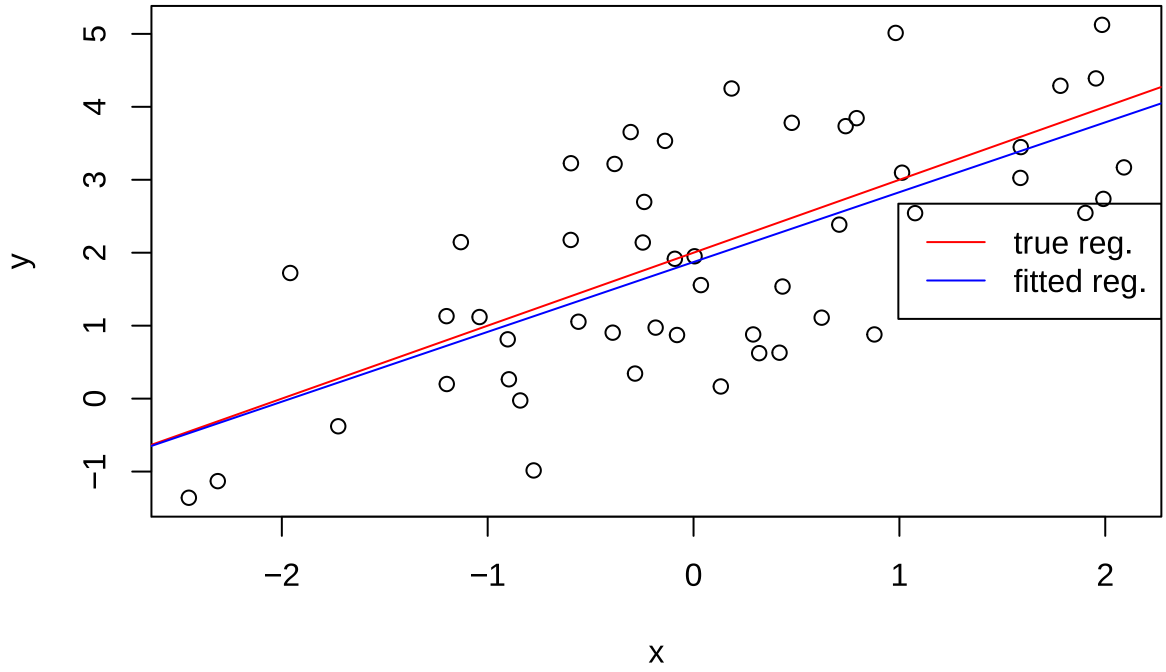
```
generate_linear <- function(n = 50, a = 1, b = 2, sigma1 = 1,
                              sigma2 = 1){
  #### first we generate data
  x <- rnorm(n, mean = 0, sd = sigma1)
  y <- a*x + b + rnorm(n, mean = 0, sd = sigma2)
  return(list(data = data.frame(x = x, y = y), a = a , b = b))
                              }
```

A function to fit the model and do the plotting

```
### this function works with a list of the type returned
### by the generate_linear function
fit_and_plot <- function(object, formula = y ~ x, verbose = TRUE){
  fit <- lm(formula = formula, data = object$data)
  if (verbose){
    plot(object$data)
    abline(a = object$b, b = object$a, col = "red")
    abline(fit, col = "blue")
    legend("right", legend = c("true reg.", "fitted reg."),
           col = c("red", "blue"), lty = 1)
    print(summary(fit)) ##we print the summary of fit
  }
  object$fit <- fit ## here we just append the fitted model to the list
  invisible(object) ## this return a temporary invisible copy, so we
                    ## do not have too many object printed in the
                    ## console, but we can still save them
}
```

We test the two functions, by default the intercept is $b = 2$ as requested by the exercise

```
set.seed(2)
fit_and_plot(generate_linear())
```

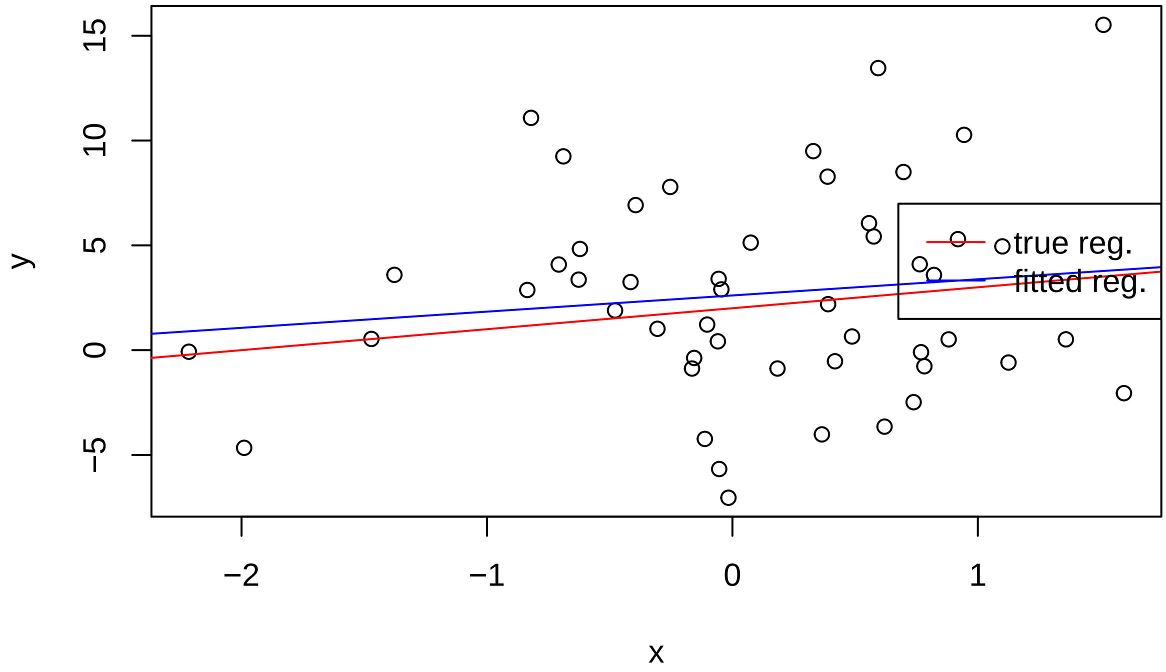


```
##
## Call:
## lm(formula = formula, data = object$data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max 
## -2.1135 -0.8608 -0.2394  1.0084  2.2018 
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept)   1.8724     0.1706  10.973 1.11e-14 ***
## x             0.9574     0.1523   6.286 9.15e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.204 on 48 degrees of freedom
## Multiple R-squared:  0.4515, Adjusted R-squared:  0.4401 
## F-statistic: 39.52 on 1 and 48 DF,  p-value: 9.145e-08
```

We can see that with the default choices $\sigma_1 = \sigma_2 = 1$, sample size $n = 50$ and $a = 1$, $b = 2$ we obtain a very strong evidence against the null hypothesis that the intercept is 0.

We try now to increase the *intrinsic noise* of $Y|X = x$, that is σ_2, (we can appreciate now the power of having defined functions)

```
set.seed(1)
fit_and_plot(generate_linear(sigma2 = 5))
```

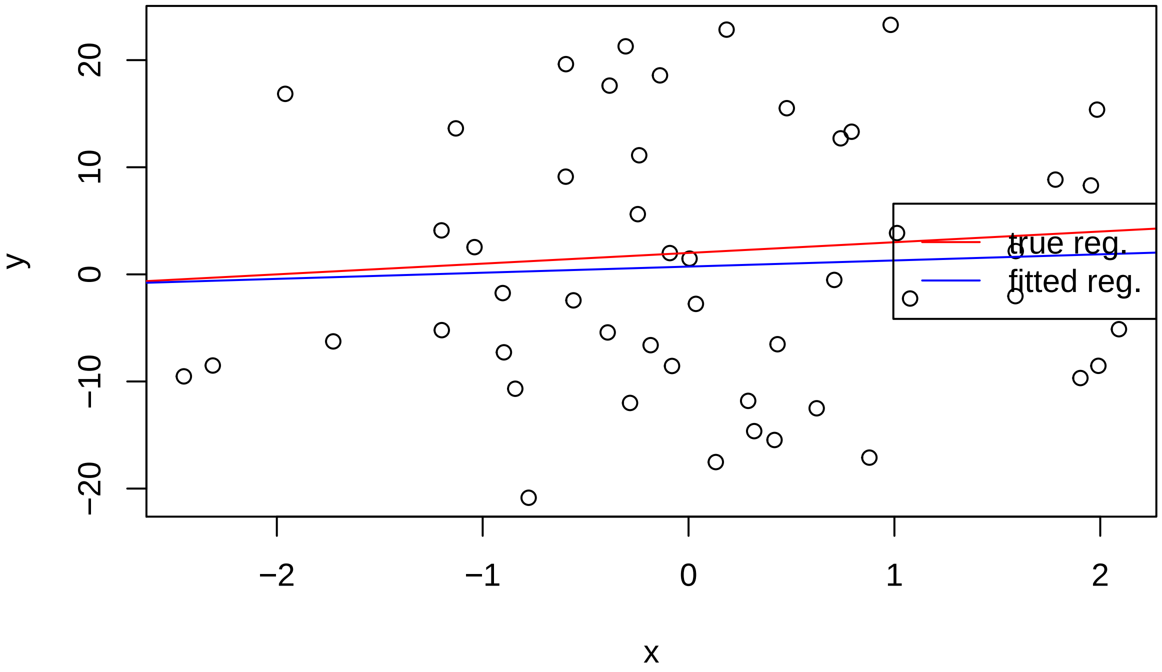


```
##
## Call:
## lm(formula = formula, data = object$data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.6380 -3.3449 -0.0112  2.4384 11.7429
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.6095     0.6968   3.745 0.000483 ***
## x             0.7723     0.8403   0.919 0.362702
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.891 on 48 degrees of freedom
## Multiple R-squared:  0.01729,    Adjusted R-squared:  -0.003183
## F-statistic: 0.8445 on 1 and 48 DF,  p-value: 0.3627
```

We can see that the p-value is reduced.

Lets increase σ_2 even more

```
set.seed(2) ### for reproducibility
fit_and_plot(generate_linear(sigma2 = 10))
```



```
##
## Call:
## lm(formula = formula, data = object$data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -21.135  -8.608  -2.394  10.084  22.018
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.7241     1.7064   0.424    0.673
## x             0.5735     1.5229   0.377    0.708
##
## Residual standard error: 12.04 on 48 degrees of freedom
## Multiple R-squared:  0.002946,   Adjusted R-squared:  -0.01783
## F-statistic: 0.1418 on 1 and 48 DF,  p-value: 0.7081
```

With this choice of σ_2 we thus no longer reject the null hypothesis. Observe that **we know** in this case that the true intercept is not 0, the problem is that the intrinsic noise on the response variable Y makes impossible to draw conclusions. We recall from the hypothesis testing theory that in this case we just can not state anything on the intercept, we can not say that the true intercept is 0!

Observe also that residual standard error is the estimator of the standard error σ_2.

You can now use the defined function to increase the sample size and check what happen.

1.5

First of all we generate some data, we could use the previous defined functions but we will do it from scratch again for clarity:

```
sigma1 <- 1
sigma2 <- 1
a <- -1
b <- 2
n <- 50
x <- rnorm(n, mean = 0, sd = sigma1)
y <- a*x + b + rnorm(n, mean = 0, sd = sigma2)
```

Then we fit both models

```
### without intercept
fit0 <- lm(formula = y ~ x - 1, data = data.frame(x = x, y = y))
### and the classical one with intercept
fit1 <- lm(formula = y ~ x, data = data.frame(x = x, y = y))
summary(fit0)
```

```
##
## Call:
## lm(formula = y ~ x - 1, data = data.frame(x = x, y = y))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.5877  1.1180  1.7084  2.5497  3.6445
##
## Coefficients:
##   Estimate Std. Error t value Pr(>|t|)
## x  -0.5742     0.2833  -2.027   0.0481 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.067 on 49 degrees of freedom
## Multiple R-squared:  0.07735,    Adjusted R-squared:  0.05852
## F-statistic: 4.108 on 1 and 49 DF,  p-value: 0.04814
```

```
summary(fit1)
```

```
##
## Call:
## lm(formula = y ~ x, data = data.frame(x = x, y = y))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.08944 -0.74320 -0.02162  0.80699  2.07733
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.8595     0.1337   13.91  < 2e-16 ***
## x            -0.8861     0.1295   -6.84 1.29e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9308 on 48 degrees of freedom
## Multiple R-squared:  0.4936, Adjusted R-squared:  0.4831
## F-statistic: 46.79 on 1 and 48 DF,  p-value: 1.293e-08
```

Obviously the model without intercept should be wrong in this case. We compare the models with AIC and BIC.

```
sapply(list(fit0 = fit0, fit1 = fit1), function(m){
  return(list(AIC = AIC(m), BIC = BIC(m)))
})
```

```
##     fit0     fit1
## AIC 217.4812 138.6824
## BIC 221.3053 144.4184
```

We can see that for both AIC and BIC the lower values are obtained by the `fit1` model, that is the model with intercept.

To perform the F-test we use the `anova` function.

```
anova(fit0, fit1)
```

```
## Analysis of Variance Table
##
## Model 1: y ~ x - 1
## Model 2: y ~ x
##   Res.Df     RSS Df Sum of Sq      F    Pr(>F)
## 1     49 209.300
## 2     48  41.587  1    167.71 193.57 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The p-value is very small and thus we have a very strong evidence against the null hypothesis that the model `fit0` (without intercept) is sufficient to describe the data. We remind that the F-test is just an exact version of the likelihood-ratio test and can be applied to nested models.

We can also repeat the experiment with a true intercept set to 0. We also increase the sample size to 300.

```
n <- 300
b <- 0
x <- rnorm(n, mean = 0, sd = sigma1)
y <- a*x + b + rnorm(n, mean = 0, sd = sigma2)
fit0 <- lm(formula = y ~ x - 1, data = data.frame(x = x, y = y))
fit1 <- lm(formula = y ~ x, data = data.frame(x = x, y = y))
sapply(list(fit0 = fit0, fit1 = fit1), function(m){
  return(list(AIC = AIC(m), BIC = BIC(m)))
})
```

```
##     fit0     fit1
## AIC 851.4144 849.9622
## BIC 858.8219 861.0735
```

```
anova(fit0, fit1)
```

```
## Analysis of Variance Table
##
## Model 1: y ~ x - 1
## Model 2: y ~ x
##   Res.Df    RSS Df Sum of Sq     F  Pr(>F)
## 1    299 296.08
## 2    298 292.69  1    3.3875 3.449 0.06428 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

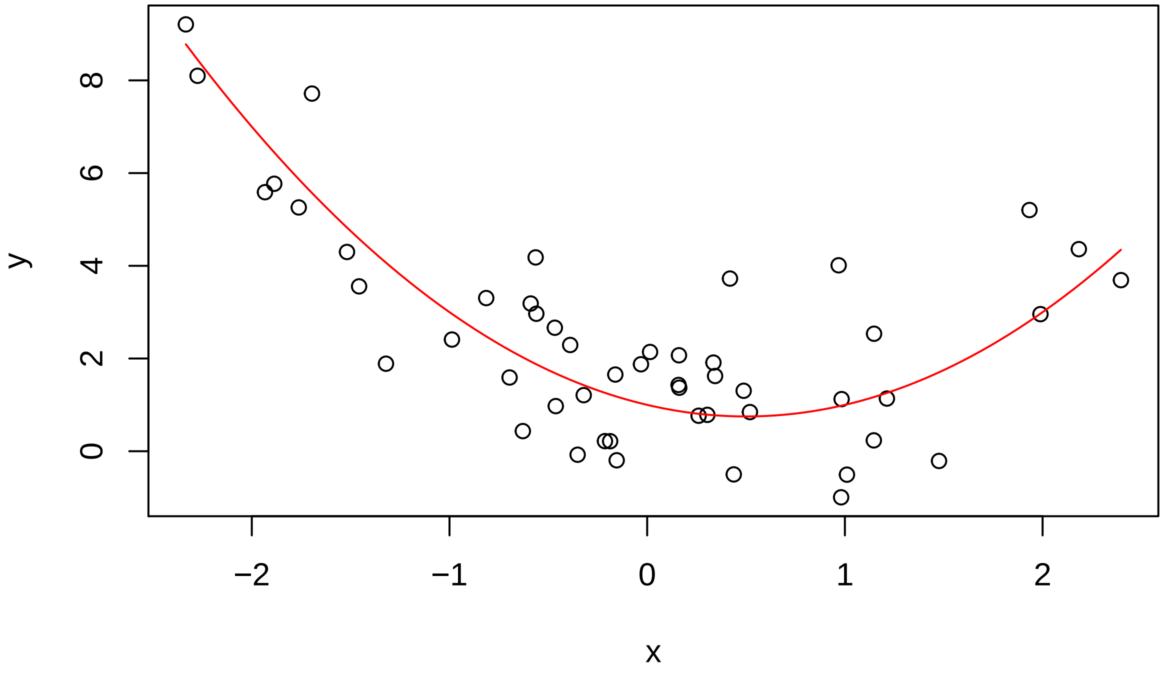
In this case we can see that the simpler model starts to be preferred, especially from the F-test p-value.

Ex 2

Here we look at polynomial models and regression. We will not go as much as in detail as in the previous exercise, and we will mainly just write the needed code.

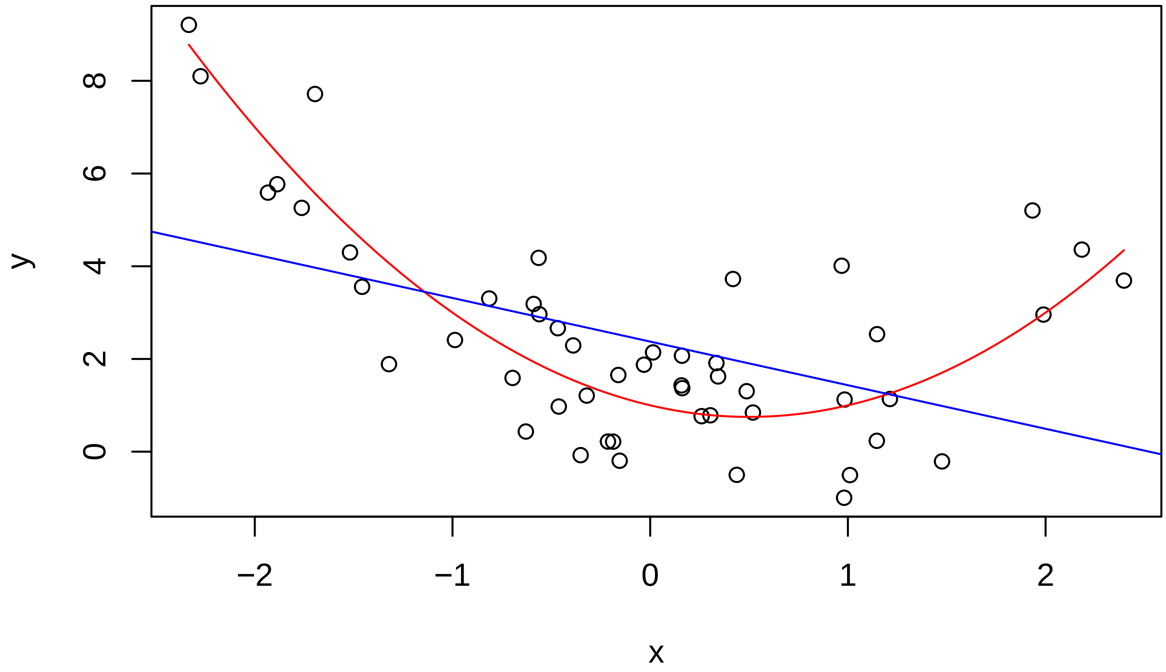
2.1

```
n <- 50
x <- rnorm(n)
r <- function(x){
  x^2 - x + 1
}
y <- r(x) + rnorm(n, sd = sqrt(2))
plot(x,y)
curve(r(x), add = T, col = "red")
```



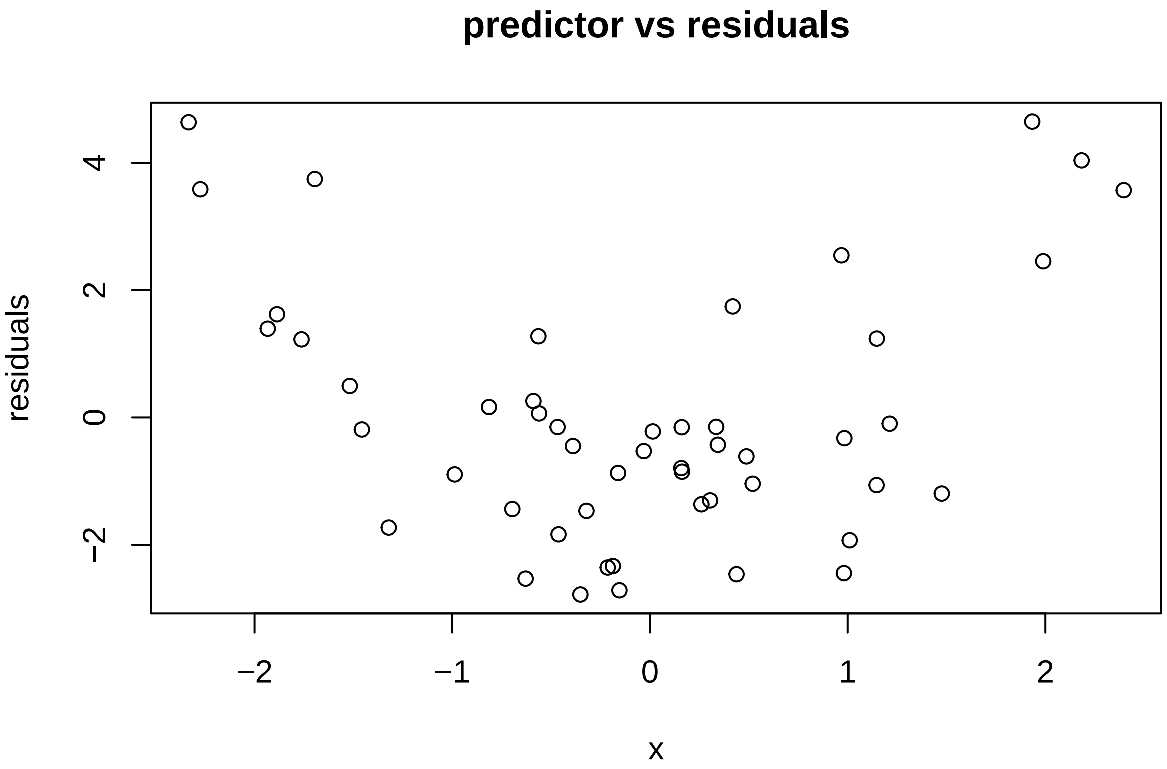
2.2

```
fit1 <- lm(y ~ x, data = data.frame(x = x, y = y))
plot(x,y)
curve(r(x), add = T, col = "red")
abline(fit1, col = "blue")
```



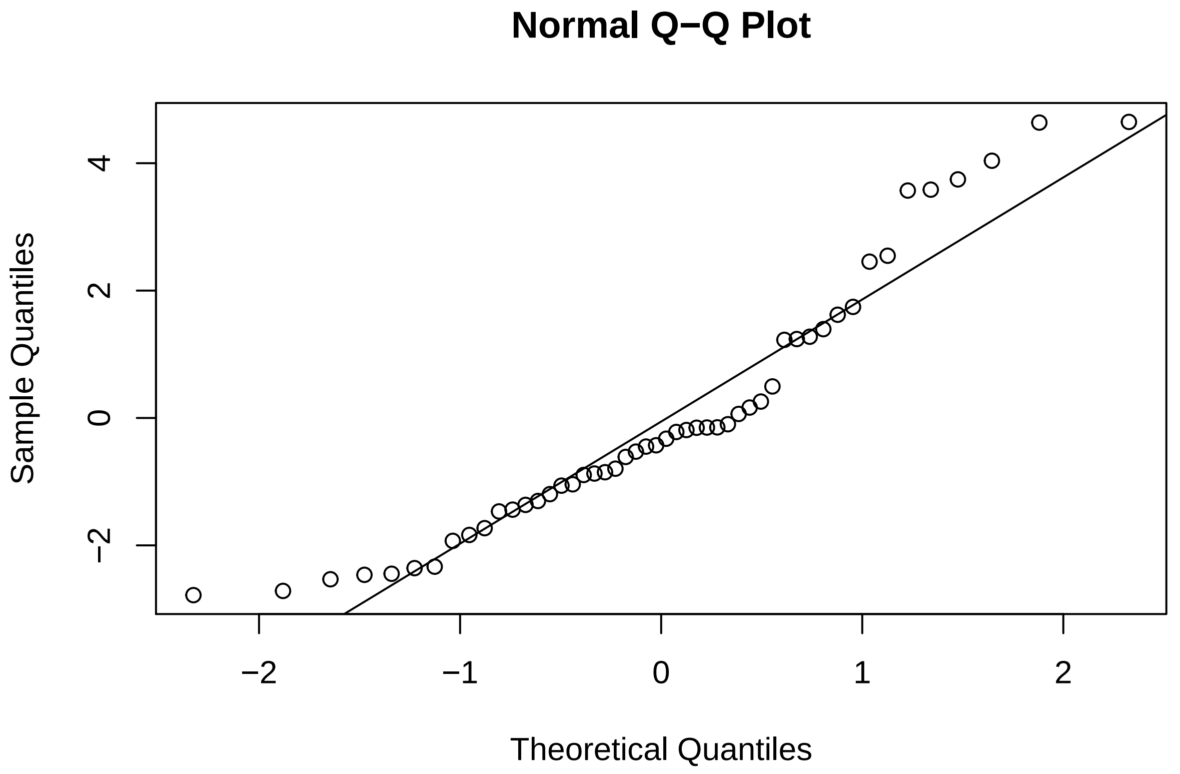
2.3

```
plot(x, fit1$residuals, ylab = "residuals",
     main = "predictor vs residuals")
```



The predictor vs residuals plot shows a clear trend, in particular the residuals do not seem independent from the predictor and not distributed around 0.

```
qqnorm(residuals(fit1))
qqline(residuals(fit1))
```

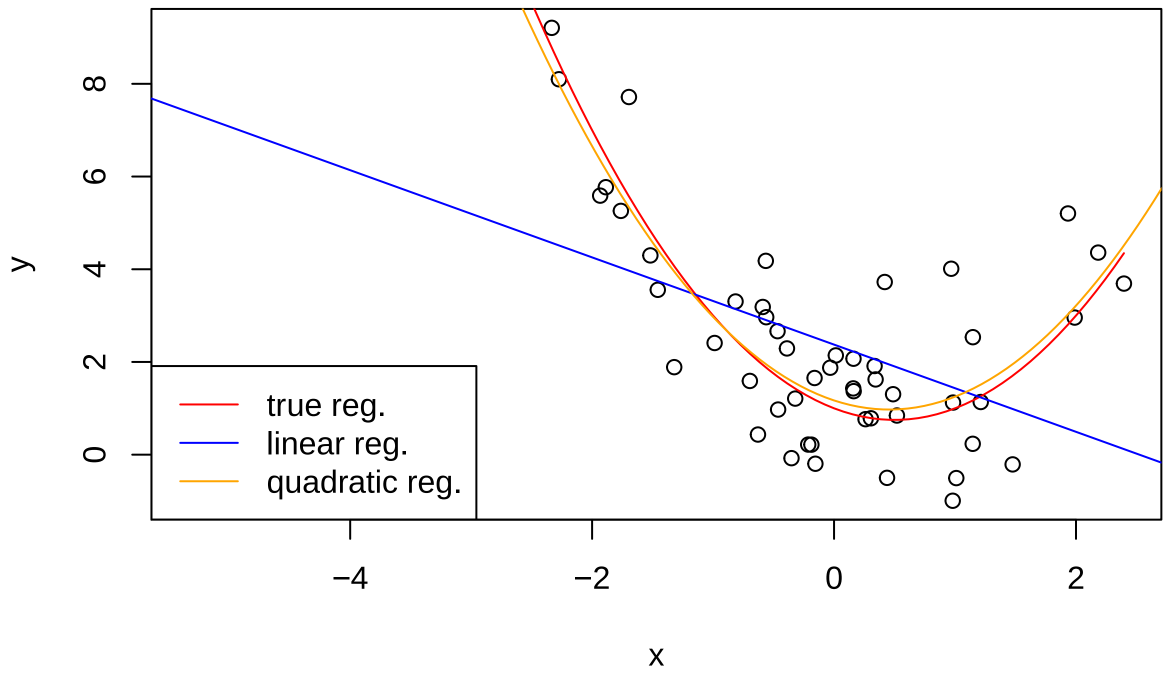


The Q-Q plot maybe is less conclusive but we can observe that the empirical quantiles of the residuals do not fit well the normal distribution.

2.4

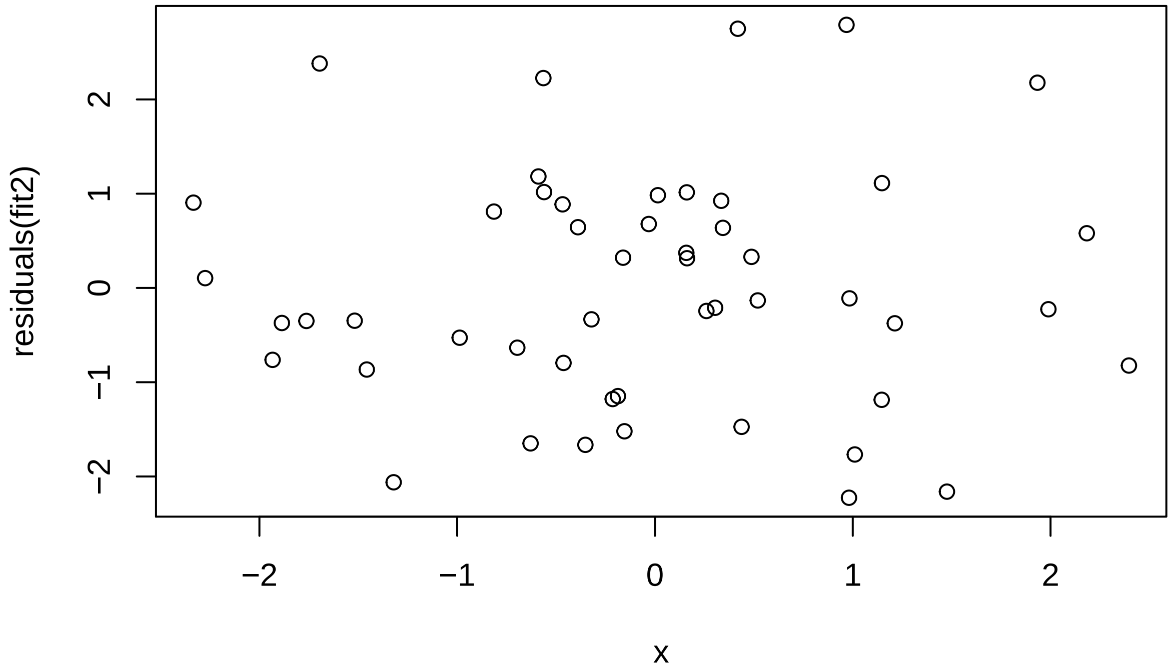
We fit now the true model, that is a polynomial of degree 2.

```r
fit2 <- lm(y ~ I(x^2) + x, data = data.frame(x = x, y = y))
plot(x,y, xlim = range(x) + c(-3, 0))
curve(r(x), add = T, col = "red")
abline(fit1, col = "blue")
xx <- seq(from = min(x) - 1, to = max(x) + 1, length.out = 100)
yy <- predict(fit2, newdata = data.frame(x = xx))
lines(xx, yy, type = "l", col = "orange")
legend("bottomleft", legend = c("true reg.", "linear reg.",
                                "quadratic reg."),
       col = c("red", "blue", "orange"), lty = 1)
```

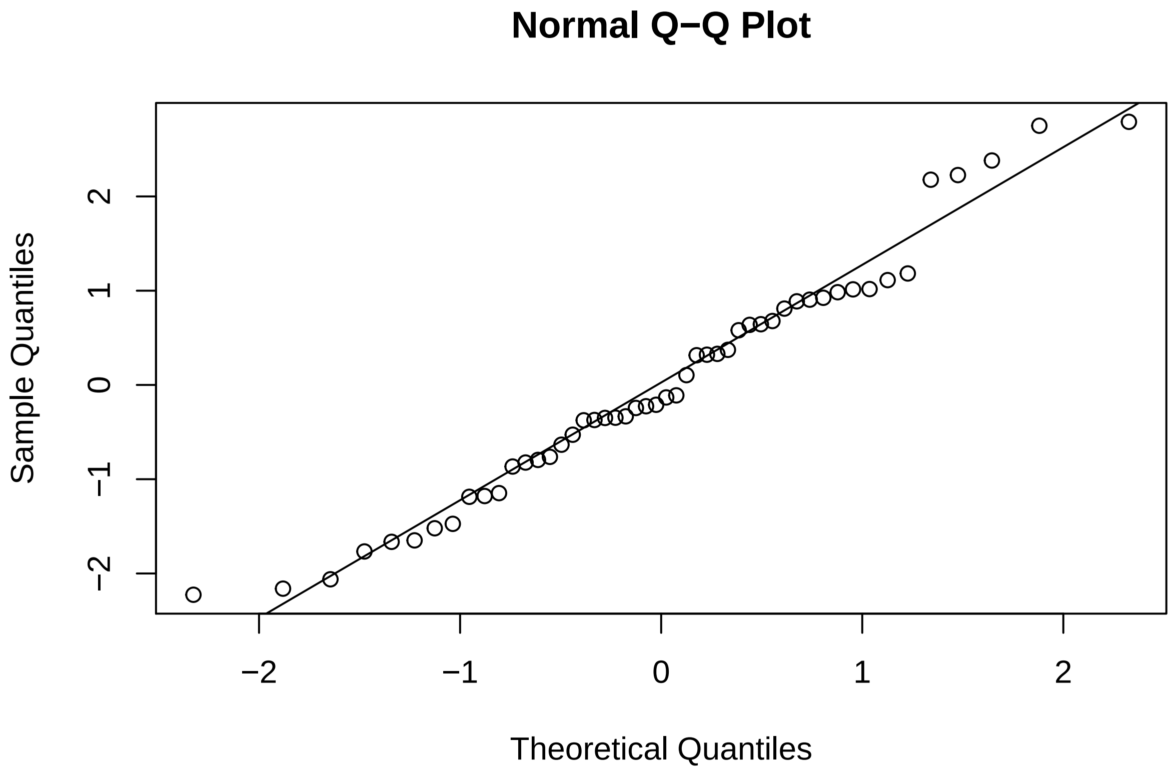


2.5

```
plot(x, residuals(fit2))
```



```r
qqnorm(residuals(fit2))
qqline(residuals(fit2))
```



Especially from the predictor vs residual plot we can observe that the residuals behaves as independent from the feature as we expect.

2.6

AIC, BIC:

```
sapply(list(fit1 = fit1, fit2 = fit2), function(m){
  return(list(AIC = AIC(m), BIC = BIC(m)))
})
```

```
##     fit1     fit2
## AIC 215.8092 171.3516
## BIC 221.5453 178.9997
```

And F-test:

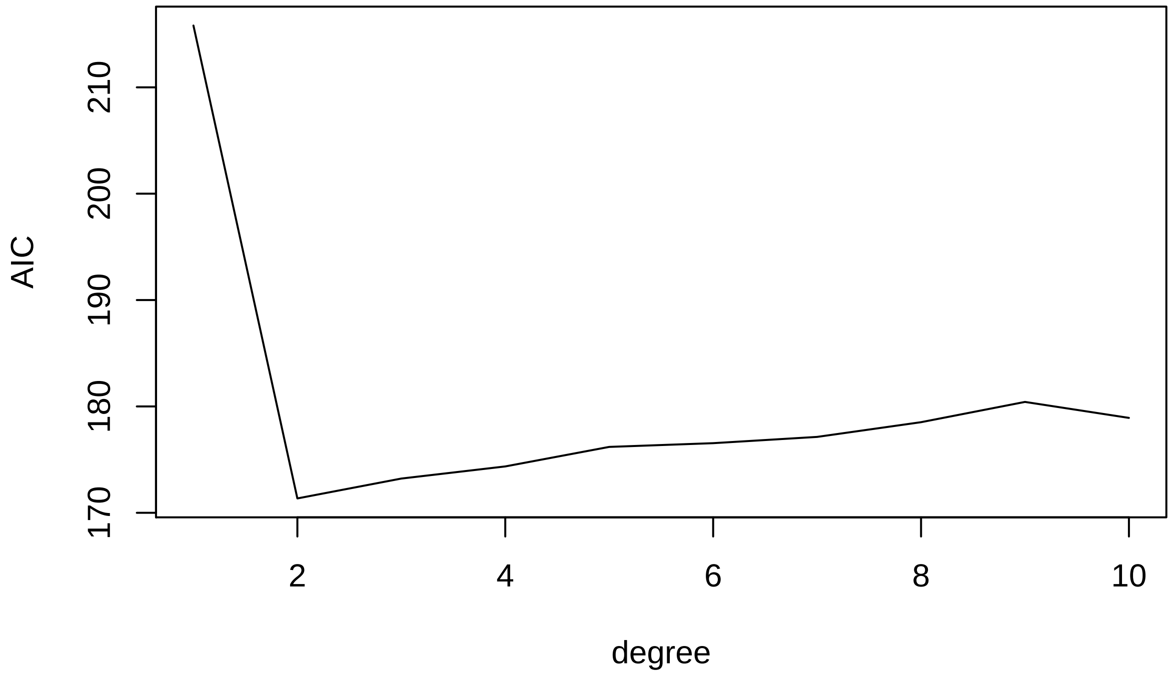
```
anova(fit1, fit2)
```

```
## Analysis of Variance Table
##
## Model 1: y ~ x
## Model 2: y ~ I(x^2) + x
##   Res.Df     RSS Df Sum of Sq      F    Pr(>F)
## 1     48 194.480
## 2     47  76.798  1    117.68 72.021 4.833e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

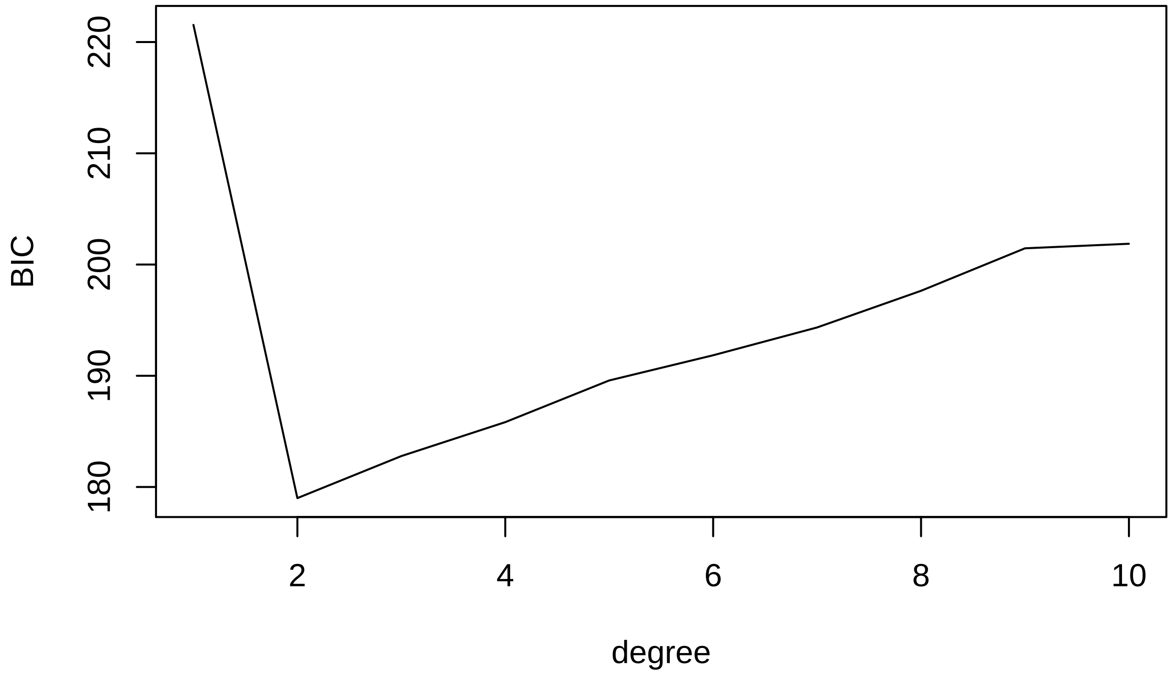
AIC/BIC and F-test both select the more complex model `fit2` (quadratic).

2.7

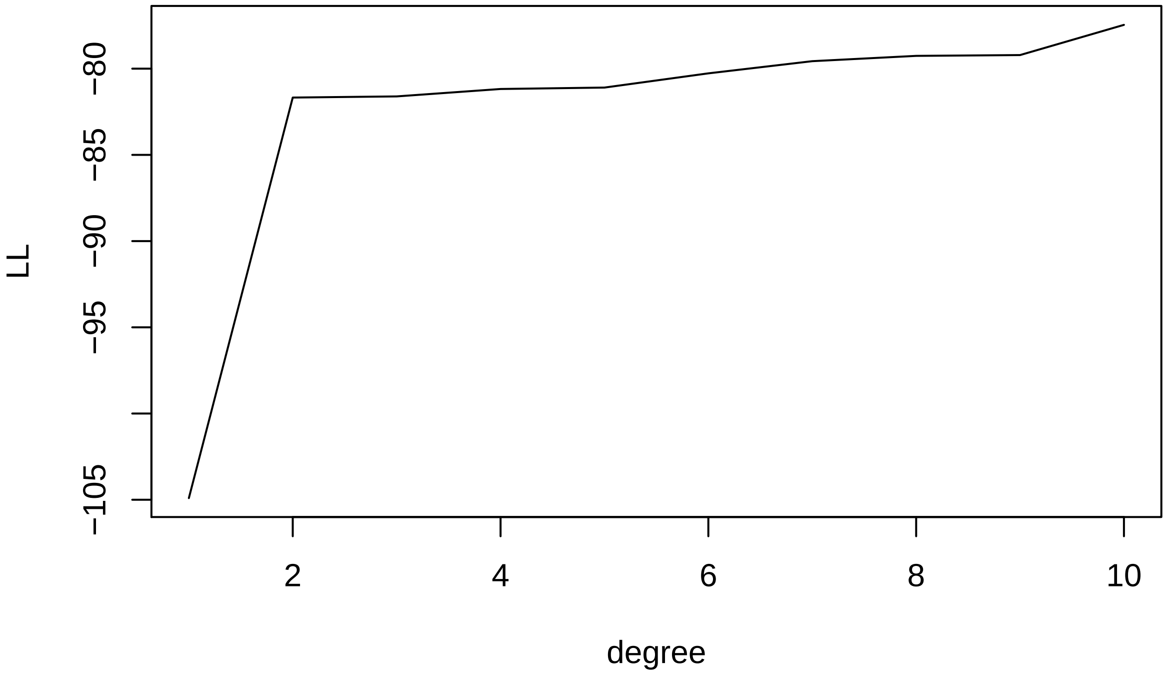
```
scores <- sapply(1:10, function(deg){
  fit <- lm(y ~ poly(x, degree = deg), data =
              data.frame(x = x, y = y))
  return(c(AIC = AIC(fit), BIC= BIC(fit), LL = logLik(fit)))
})
colnames(scores) <- 1:10
plot(scores[1,], type = "l", xlab = "degree", ylab = "AIC" )
```



```
plot(scores[2,], type = "l", xlab = "degree", ylab = "BIC" )
```



```
plot(scores[3,], type = "l", xlab = "degree", ylab = "LL" )
```

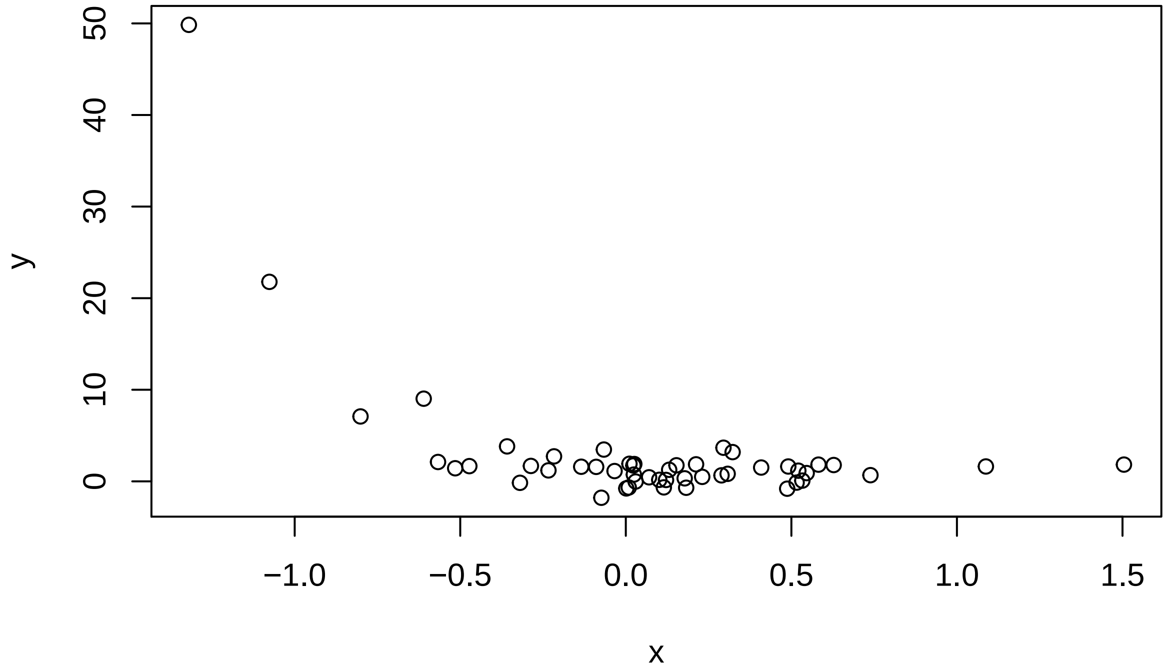


We can see that the log likelihood is an increasing function of the degree of the polynomial (or in general of the number of parameters). That is, the more parameters we have in the model, the better it will fit a given data set. That is why we can not use the pure likelihood for model selection because we will always choose the more complex model. AIC and BIC are two ways of penalizing the complexity of the model, so that we choose the more complex model only if it obtain a **significant** increase in the log-likelihood.

Ex 3

3.1

```
n <- 50
x <- rnorm(n, sd = 1/2)
r <- function(x){
  exp(-3*x) + 2*x
}
y <- r(x) + rnorm(n, sd = sqrt(2))
plot(x,y)
```

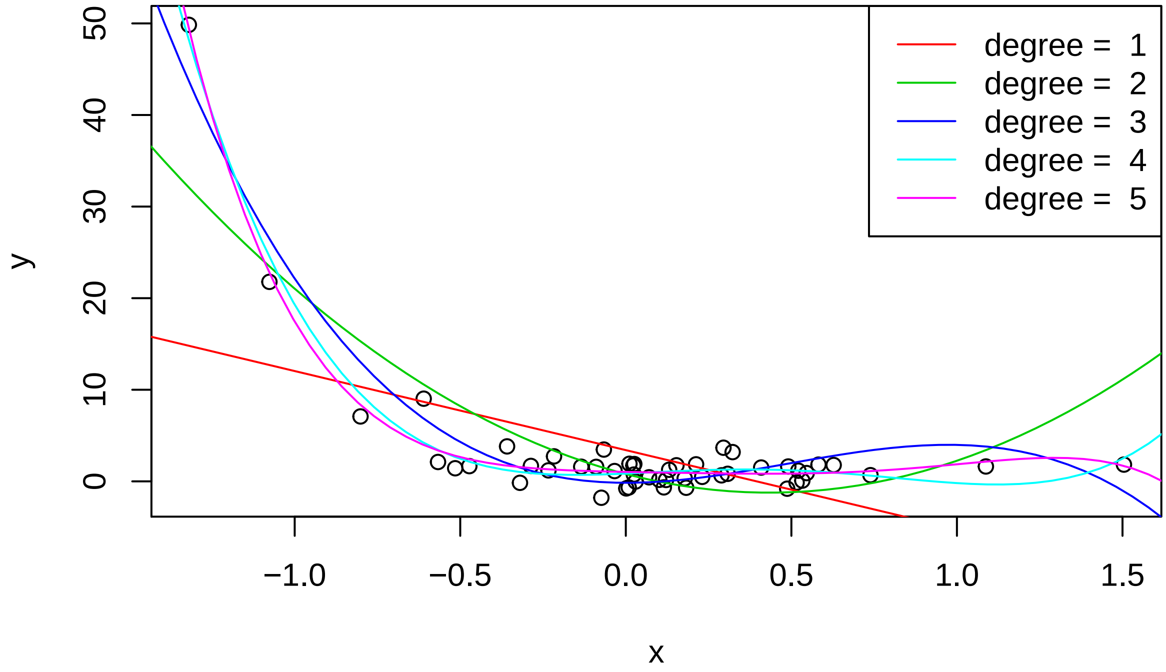


3.2

```
candidates <- lapply(1:5, function(d){
  lm(y ~ poly(x, degree = d), data = data.frame(x = x, y = y))
})
```

We can also plot all the fitted regression functions

```
plot(x,y)
xx <- seq(from = min(x) -1, to = max(x) + 1, length.out = 100)
newdata <- data.frame(x = xx)
invisible(lapply(candidates, function(m){
  k <- length(m$coefficients)
  lines(xx, predict(m, newdata = newdata), col = k)
  }))
legend("topright", legend = paste("degree = ", 1:5),
       col = 2:6, lty = 1)
```



3.3

```
sapply(candidates, function(m){
  return(list(AIC = AIC(m), BIC = BIC(m)))
})
```

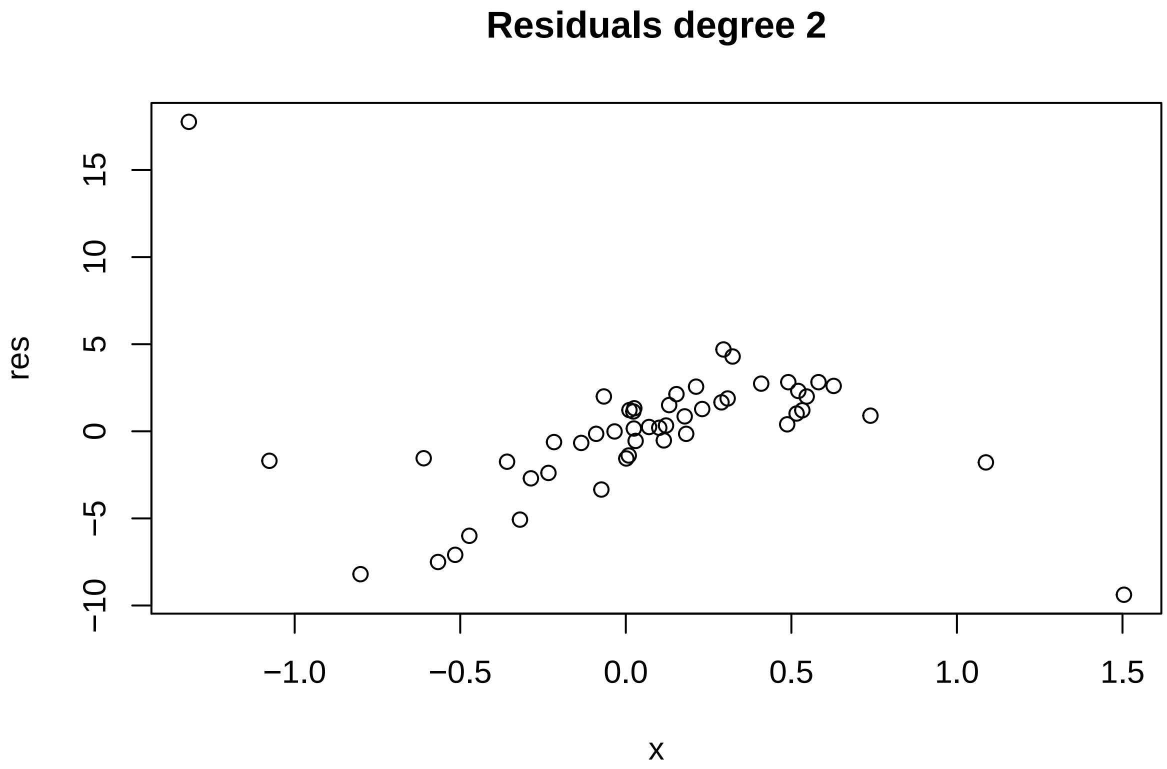
```
##     [,1]     [,2]     [,3]     [,4]     [,5]
## AIC 330.6953 288.2006 229.9735 194.6724 188.2364
## BIC 336.4314 295.8487 239.5336 206.1446 201.6206
```

Using AIC we select the 5 degree polynomial model and with BIC the 4 degree polynomial.

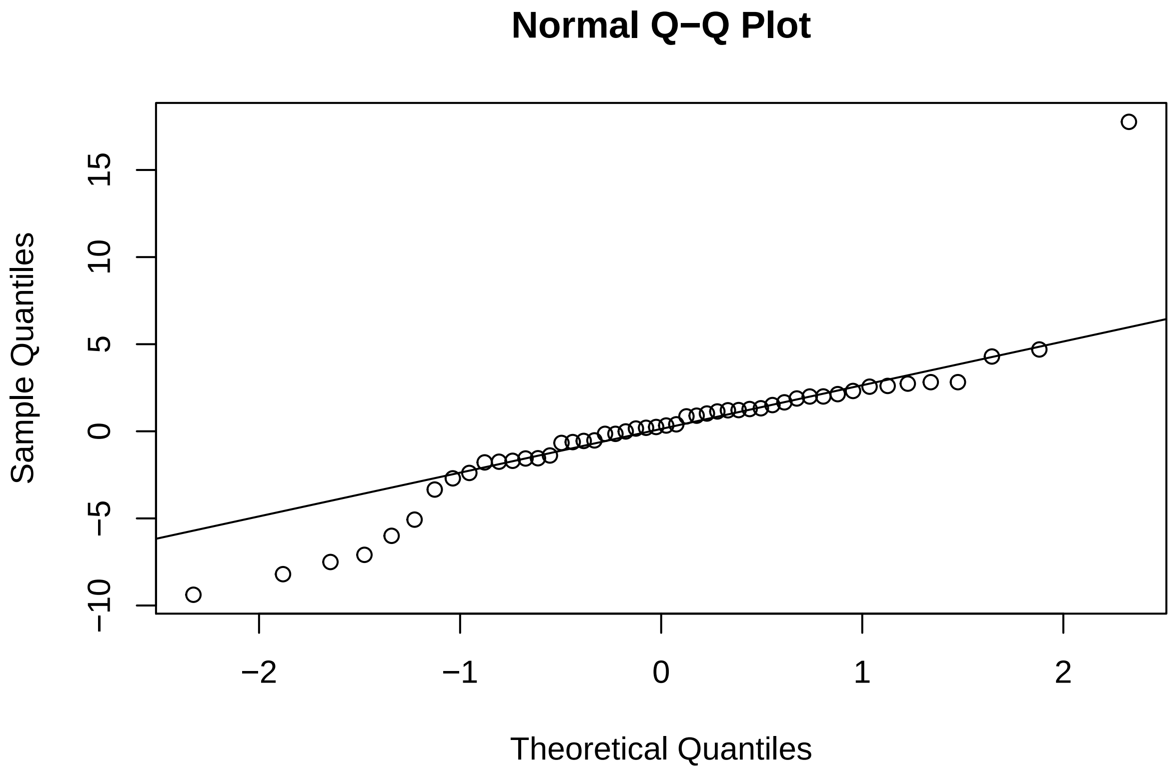
3.4

We plot residuals for the 2 degree polynomial.

```
res <- residuals(candidates[[2]])
plot(x, res, main = "Residuals degree 2")
```

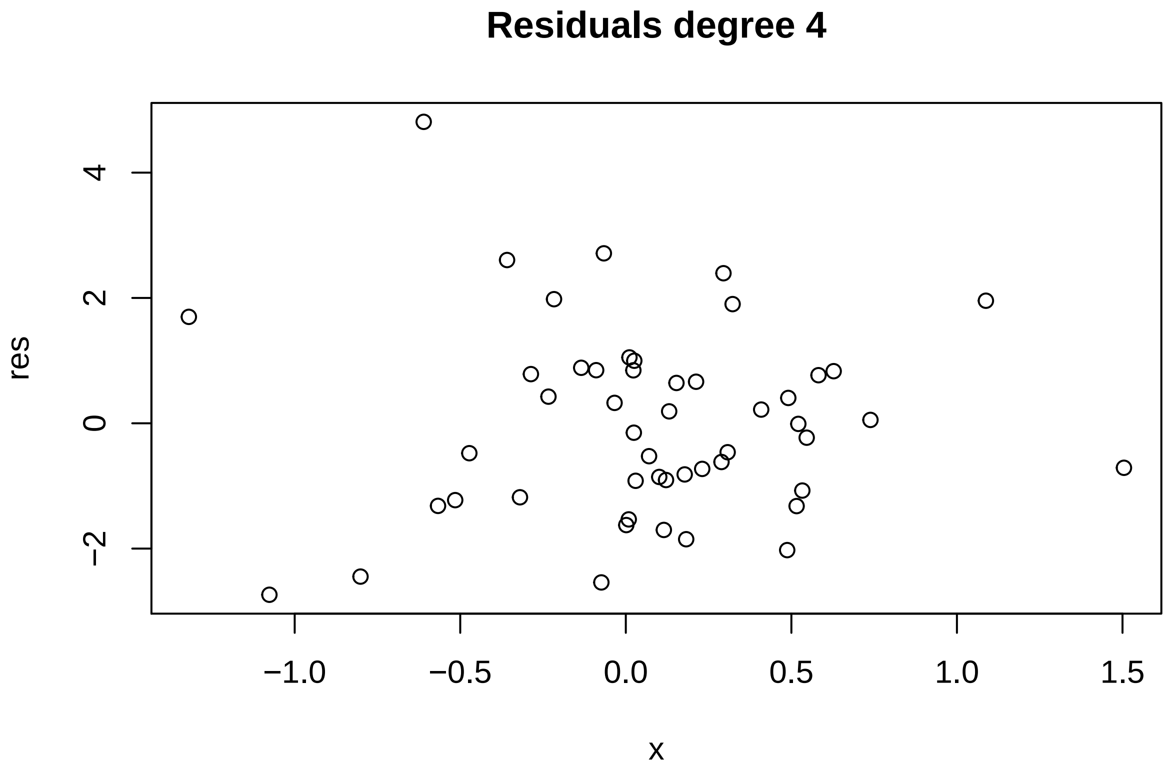


```r
qqnorm(res)
qqline(res)
```

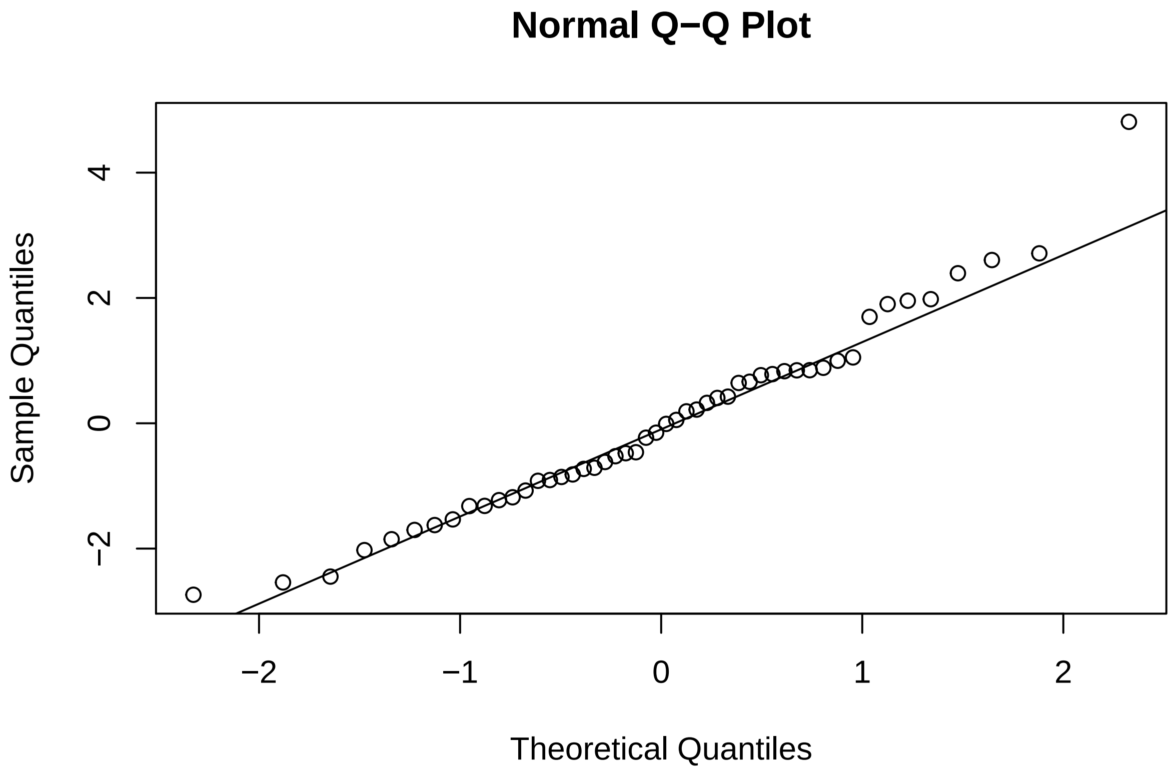


We plot residuals for the 4 degree polynomial.

```
res <- residuals(candidates[[4]])
plot(x, res, main = "Residuals degree 4")
```



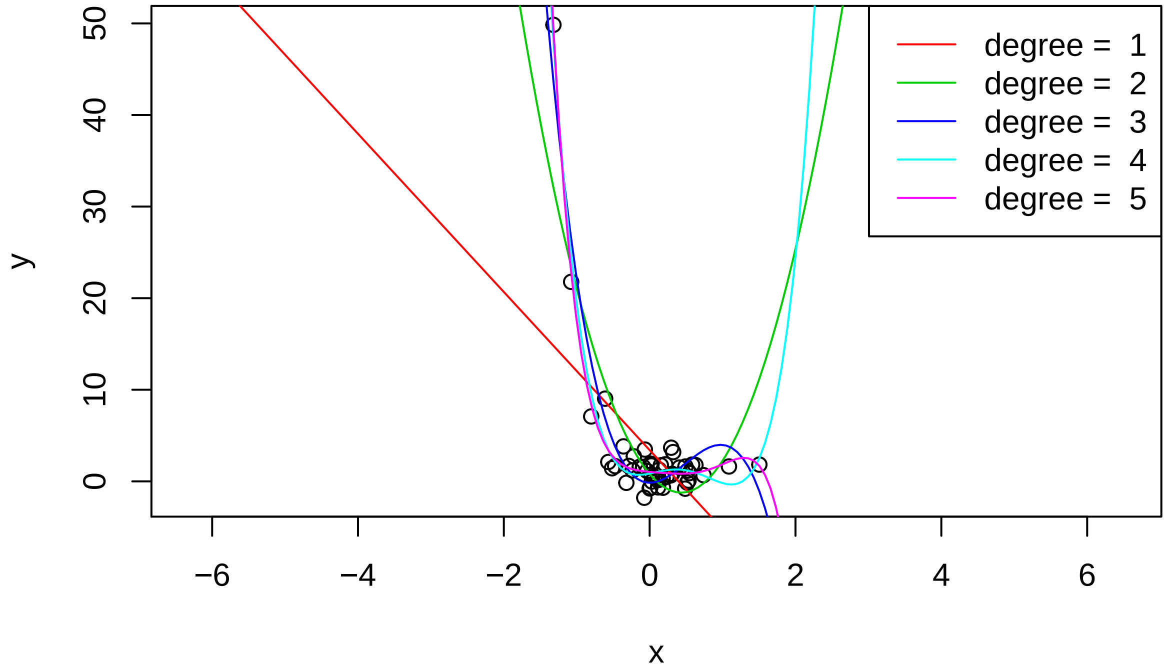
```
qqnorm(res)
qqline(res)
```



From the residuals scatter plot and the normal Q-Q plot it is difficult to tell if the model are correct in this case. We know that the true regression function is not polynomial.

We can plot the behavior of the regression functions outside the range of the data.

```
plot(x,y, xlim = range(x) + c(-5, 5))
xx <- seq(from = min(x) - 10 , to = max(x) + 10, length.out = 300)
newdata <- data.frame(x = xx)
invisible(lapply(candidates, function(m){
  k <- length(m$coefficients)
  lines(xx, predict(m, newdata = newdata), col = k)
}))
legend("topright", legend = paste("degree = ", 1:5),
       col = 2:6, lty = 1)
```



We can observe that depending on the degree of the polynomial used the models behave very differently, especially outside of the range of the data.

3.5

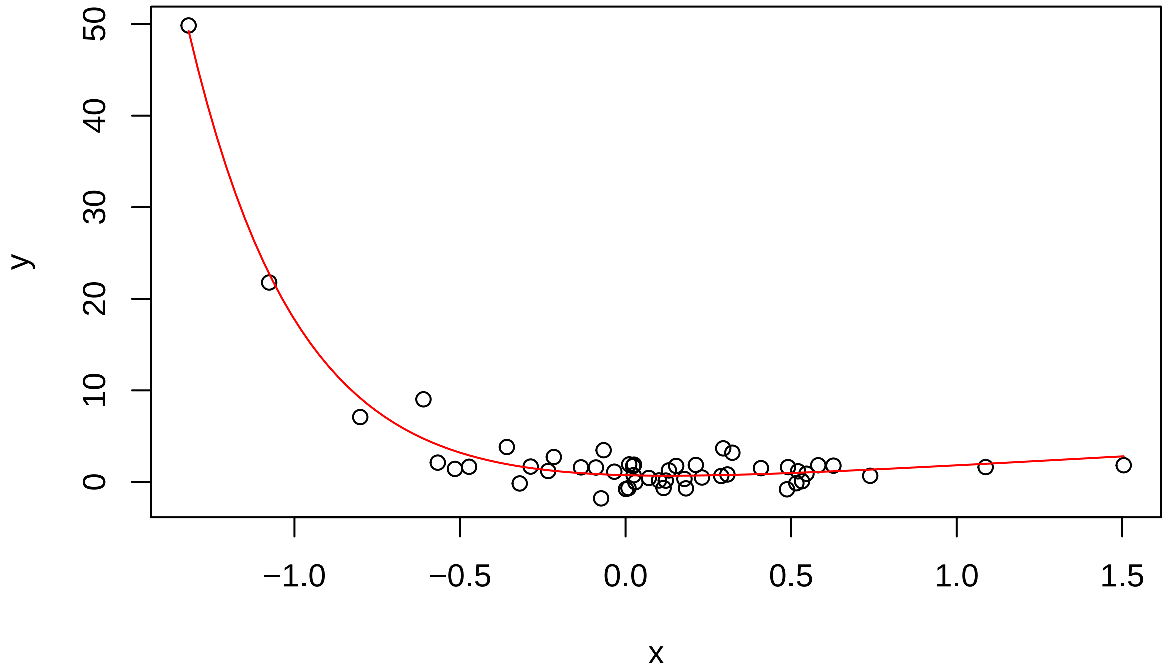
We now fit the real model

```r
regr <- function(x, b){
  b[1] + b[2]*x + exp(b[3] * x)
}

rss <- function(b){
  sum( (y - regr(x, b))^2 )
}


result <- optim(par = c(0, -1, -1), fn = rss)
result
```

```
## $par
## [1] -0.2586232  2.0281709 -2.9970268
##
## $value
## [1] 96.25826
##
## $counts
## function gradient
##      162       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

We can now plot the obtained regression function

```
plot(x,y)
curve(regr(x, result$par), add = TRUE, col = "red")
```



The built-in method for non-linear least squares obtains the same values for the parameters:

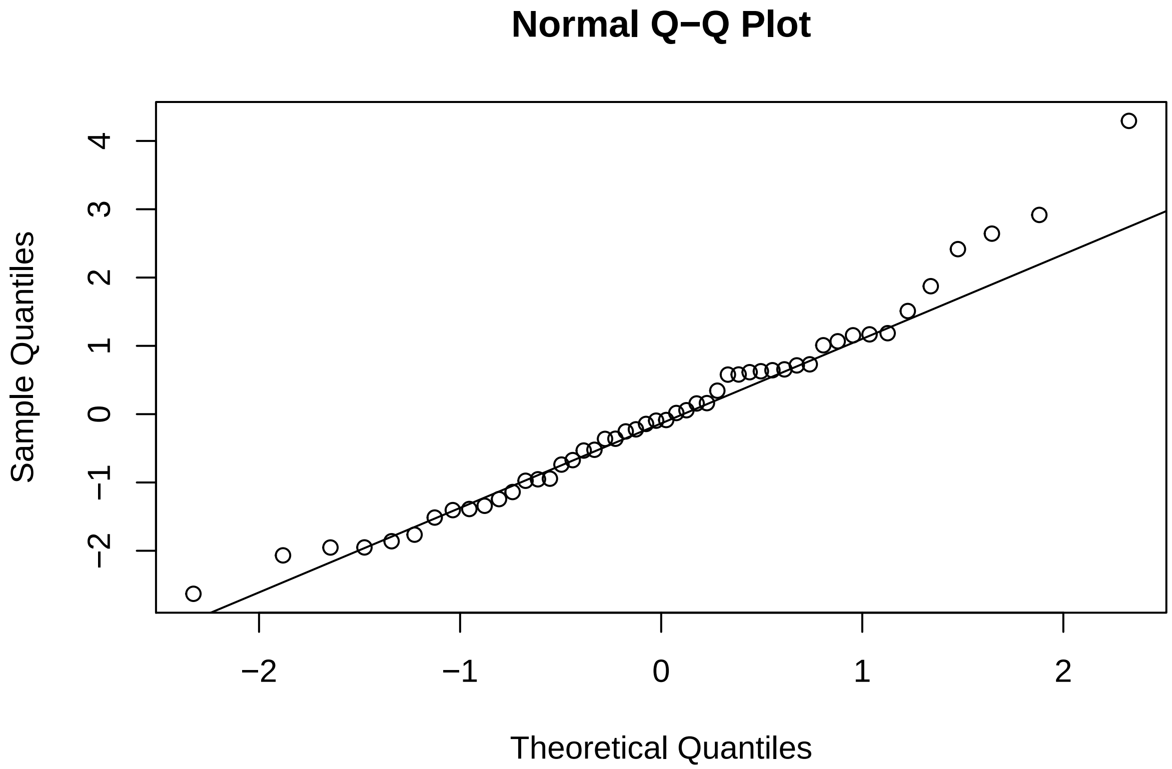
```
fitnls <- nls(formula = y ~ b0 + b1*x + exp(b2*x), start = list(b0 = 0, b1 = 0, b2 = -5),
              data = data.frame(x = x, y = y))
summary(fitnls)
```

```
##
## Formula: y ~ b0 + b1 * x + exp(b2 * x)
##
## Parameters:
##    Estimate Std. Error  t value Pr(>|t|)
## b0 -0.25886    0.21879   -1.183   0.2427
## b1  2.02805    0.50264    4.035   0.0002 ***
## b2 -2.99700    0.02389 -125.463   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.431 on 47 degrees of freedom
##
## Number of iterations to convergence: 7
## Achieved convergence tolerance: 8.507e-08
```

The algorithms are very sensitive to the starting points, especially if we start from a positive value for the `b2` parameter.

We can now check the residuals of the model.

```
qqnorm(residuals(fitnls))
qqline(residuals(fitnls))
```



We can also fit the model maximizing the log-likelihood (or minimizing the minus log-likelihood)

```
mll <- function(pars){
  - sum( sapply(1:n, function(i){
    return(dnorm(y[i],
                 mean = regr(x[i],
                             pars[1:3]),
                 sd = pars[4], log = T))
  }))
}

optim(mll, par = c(0,0,-5,1))
```

```
## $par
## [1] -0.2505729  2.0180609 -2.9972748  1.3915197
##
## $value
## [1] 87.32412
##
## $counts
## function gradient
##      169       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

The last parameter is the estimated value of the noise standard deviation.

The wine quality dataset

We load the data with

```
wines <- read.csv("winequality-red.csv", sep =";")
```

Ex 4

4.1

```
fitall <- lm(quality ~ ., data = wines)
summary(fitall)
```

```
##
## Call:
## lm(formula = quality ~ ., data = wines)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.68911 -0.36652 -0.04699  0.45202  2.02498
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)           2.197e+01  2.119e+01   1.036   0.3002
## fixed.acidity         2.499e-02  2.595e-02   0.963   0.3357
## volatile.acidity     -1.084e+00  1.211e-01  -8.948  < 2e-16 ***
## citric.acid          -1.826e-01  1.472e-01  -1.240   0.2150
## residual.sugar        1.633e-02  1.500e-02   1.089   0.2765
## chlorides            -1.874e+00  4.193e-01  -4.470 8.37e-06 ***
## free.sulfur.dioxide   4.361e-03  2.171e-03   2.009   0.0447 *
## total.sulfur.dioxide -3.265e-03  7.287e-04  -4.480 8.00e-06 ***
## density              -1.788e+01  2.163e+01  -0.827   0.4086
## pH                   -4.137e-01  1.916e-01  -2.159   0.0310 *
## sulphates             9.163e-01  1.143e-01   8.014 2.13e-15 ***
## alcohol               2.762e-01  2.648e-02  10.429  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 0.648 on 1587 degrees of freedom
## Multiple R-squared:  0.3606, Adjusted R-squared:  0.3561
## F-statistic: 81.35 on 11 and 1587 DF,  p-value: < 2.2e-16
```

From the p-values of the t-tests we can observe that probably the most significant regressors are `volatile.acidity`, `chlorides`, `total.sulfur.dioxide`, `sulphates`, `alcohol`, `ph` and `free.sulfur.dioxide`.

4.2

We implement now the forward stepwise selection (with AIC),

```
fit <- lm(quality ~ 1, data = wines) ## only the intercept
regressors <- colnames(wines)[-12]
selected <- c()
score <- AIC(fit)
score.best <- score
done <- FALSE
while (!done){
   for (reg in regressors[!(regressors %in% selected)]){
     tmp <- update(fit, formula = paste(". ~ . + ", reg))
     score.tmp <- AIC(tmp)
     if (score.tmp < score.best){
       score.best <- score.tmp
       best <- tmp
       selected.best <- c(selected, reg)
     }
   }
   if (score.best < score){
     fit <- best
     score <- score.best
     selected <- selected.best
   }else{ ### if there is no increase
     done <- TRUE
   }
}
#### when the while loop ends we will have the selected model in
#### fit

fit
```

```
##
## Call:
## lm(formula = quality ~ alcohol + volatile.acidity + sulphates +
##     total.sulfur.dioxide + chlorides + pH + free.sulfur.dioxide,
##     data = wines)
##
## Coefficients:
##          (Intercept)               alcohol      volatile.acidity
##             4.430099              0.289303             -1.012753
##            sulphates  total.sulfur.dioxide             chlorides
##             0.882665             -0.003482             -2.017814
##                   pH   free.sulfur.dioxide
##            -0.482661              0.005077
```

Using the built-in function `step` we obtain the same result,

```
fit <- lm(quality ~ 1, data = wines) ## only the intercept
biggest <- lm(quality ~ ., data = wines)
fit_step <- step(object = fit, scope = formula(biggest), direction = "forward",
                 trace = 0)
fit_step
```

```
##
## Call:
## lm(formula = quality ~ alcohol + volatile.acidity + sulphates +
##     total.sulfur.dioxide + chlorides + pH + free.sulfur.dioxide,
##     data = wines)
##
## Coefficients:
##          (Intercept)               alcohol      volatile.acidity
##             4.430099              0.289303             -1.012753
##            sulphates  total.sulfur.dioxide             chlorides
##             0.882665             -0.003482             -2.017814
##                   pH   free.sulfur.dioxide
##            -0.482661              0.005077
```

4.3

We now implement the method of Zheng-Loh for model selection.

First of all we fit the full model and we obtain the Wald statistics. Then we sort the absolute values of the Wald statistics.

```
st.errors <- summary(fitall)$coefficients[-1,2]
W <- fitall$coefficients[-1] / st.errors
ix <- sort(abs(W), decreasing = TRUE, index.return = TRUE)$ix
reg.names <- names(fitall$coefficients[-1])[ix] ### sorted
sigma2_est <- sum(fitall$residuals^2) / nrow(wines)
```

Now we select the model that minimize $RSS(j) + j\hat{\sigma}^2 \log(n)$.

```
s <- Inf
for (j in 1:length(reg.names)){
  fit.tmp <- lm(paste("quality ~",
                      paste(reg.names[1 : j], collapse = "+") ),
                data = wines)
  s.tmp <- sum(fit.tmp$residuals^2) + j * (sigma2_est) *
      log(nrow(wines))
  if (s.tmp < s ){
        J <- j
        s <- s.tmp
      }
}

fit.final <- lm(paste("quality ~",
                      paste(reg.names[1 : J], collapse = "+") ),
                data = wines)
summary(fit.final)
```

```
##
## Call:
## lm(formula = paste("quality ~", paste(reg.names[1:J], collapse = "+")),
##     data = wines)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.60575 -0.35883 -0.04806  0.46079  1.95643
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)           4.2957316  0.3995603  10.751  < 2e-16 ***
## alcohol               0.2906738  0.0168108  17.291  < 2e-16 ***
## volatile.acidity     -1.0381945  0.1004270 -10.338  < 2e-16 ***
## sulphates             0.8886802  0.1100419   8.076 1.31e-15 ***
## total.sulfur.dioxide -0.0023721  0.0005064  -4.684 3.05e-06 ***
## chlorides            -2.0022839  0.3980757  -5.030 5.46e-07 ***
## pH                   -0.4351830  0.1160368  -3.750 0.000183 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6487 on 1592 degrees of freedom
## Multiple R-squared:  0.3572, Adjusted R-squared:  0.3548
## F-statistic: 147.4 on 6 and 1592 DF,  p-value: < 2.2e-16
```

Ex 5

We transform the quality variable to a binary variable indicating if a wine is good or bad.

```
good <- wines$quality > 5
wines$quality <- "bad"
wines[good, "quality"] <- "good"
wines[,"quality"] <- as.factor(wines[, "quality"])
```

5.1

We fit now a logistic regression model using all predictors.

```
fitA <- glm(quality ~ . , family = "binomial",
            data = wines )
summary(fitA)
```

```
##
## Call:
## glm(formula = quality ~ ., family = "binomial", data = wines)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.4025  -0.8387   0.3105   0.8300   2.3142
##
## Coefficients:
##                        Estimate Std. Error z value Pr(>|z|)
## (Intercept)           42.949948  79.473979   0.540  0.58890
## fixed.acidity          0.135980   0.098483   1.381  0.16736
## volatile.acidity      -3.281694   0.488214  -6.722 1.79e-11 ***
## citric.acid           -1.274347   0.562730  -2.265  0.02354 *
## residual.sugar         0.055326   0.053770   1.029  0.30351
## chlorides             -3.915713   1.569298  -2.495  0.01259 *
## free.sulfur.dioxide    0.022220   0.008236   2.698  0.00698 **
## total.sulfur.dioxide  -0.016394   0.002882  -5.688 1.29e-08 ***
## density              -50.932385  81.148745  -0.628  0.53024
## pH                    -0.380608   0.720203  -0.528  0.59717
## sulphates              2.795107   0.452184   6.181 6.36e-10 ***
## alcohol                0.866822   0.104190   8.320  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2209.0  on 1598  degrees of freedom
## Residual deviance: 1655.6  on 1587  degrees of freedom
## AIC: 1679.6
##
## Number of Fisher Scoring iterations: 4
```

5.2

We implement a forward feature selection based on AIC.

```r
fit <- glm(quality ~ 1, family = "binomial",
           data = wines) ## only the intercept
regressors <- colnames(wines)[-12]
selected <- c()
score <- AIC(fit)
score.best <- score
done <- FALSE
while (!done){
   for (reg in regressors[!(regressors %in% selected)]){
     tmp <- update(fit, formula = paste(". ~ . + ", reg))
     score.tmp <- AIC(tmp)
     if (score.tmp < score.best){
       score.best <- score.tmp
       best <- tmp
       selected.best <- c(selected, reg)
     }
   }
   if (score.best < score){
     fit <- best
     score <- score.best
     selected <- selected.best
   }else{ ### if there is no increase
     done <- TRUE
   }
}
#### when the while loop ends we will have the selected model in
#### fit

fit
```

```
##
## Call:  glm(formula = quality ~ alcohol + volatile.acidity + total.sulfur.dioxide +
##     sulphates + chlorides + free.sulfur.dioxide + pH + citric.acid +
##     fixed.acidity, family = "binomial", data = wines)
##
## Coefficients:
##          (Intercept)               alcohol      volatile.acidity
##             -6.93847               0.91774              -3.30102
## total.sulfur.dioxide             sulphates             chlorides
##             -0.01622               2.70071              -3.94020
##  free.sulfur.dioxide                    pH           citric.acid
##              0.02317              -0.63567              -1.24344
##        fixed.acidity
##              0.09000
##
## Degrees of Freedom: 1598 Total (i.e. Null);  1589 Residual
## Null Deviance:       2209
## Residual Deviance: 1657  AIC: 1677
```

Using log-likelihood

```
fit_ll <- glm(quality ~ 1, family = "binomial",
           data = wines) ## only the intercept
regressors <- colnames(wines)[-12]
selected <- c()
score <- -logLik(fit_ll)
score.best <- score
done <- FALSE
while (!done){
   for (reg in regressors[!(regressors %in% selected)]){
     tmp <- update(fit, formula = paste(". ~ . + ", reg))
     score.tmp <- -logLik(tmp)
     if (score.tmp < score.best){
       score.best <- score.tmp
       best <- tmp
       selected.best <- c(selected, reg)
     }
   }
   if (score.best < score){
     fit_ll <- best
     score <- score.best
     selected <- selected.best
   }else{ ### if there is no increase
     done <- TRUE
   }
}
#### when the while loop ends we will have the selected model in
#### fit

fit_ll
```

```
##
## Call:  glm(formula = quality ~ alcohol + volatile.acidity + total.sulfur.dioxide +
##     sulphates + chlorides + free.sulfur.dioxide + pH + citric.acid +
##     fixed.acidity + residual.sugar, family = "binomial", data = wines)
##
## Coefficients:
##          (Intercept)               alcohol      volatile.acidity
##             -6.91333               0.91268              -3.32573
## total.sulfur.dioxide             sulphates             chlorides
##             -0.01654               2.72654              -4.00691
##  free.sulfur.dioxide                    pH           citric.acid
##              0.02265              -0.63906              -1.27836
##        fixed.acidity        residual.sugar
##              0.08787               0.03531
##
## Degrees of Freedom: 1598 Total (i.e. Null);  1588 Residual
## Null Deviance:       2209
## Residual Deviance: 1656  AIC: 1678
```

5.3

Using log-likelihood to select the model we will always select the more complex model, since more complex models will always fit better the data than simpler ones.

That is why we need to penalize complexity as in AIC or BIC.

5.4

```
preds <- predict(fit)
linkinv <- binomial()$linkinv

## From the doc of binomial() :
## As a factor: ‘success’ is interpreted
## as the factor not having the first
##level (and hence usually of having the
##second level).
linkinv(preds)[1] ##prob of success
```

```
##         1
## 0.2190518
```

```
toClass <- function(predictions,
                    levels,
                    linkinv =
                    binomial()$linkinv){

  ## threshold the prob of success
  a <- linkinv(predictions) > 0.5
  b <- array(dim =
               c(length(predictions)))
  ## if prob succ > 0.5 => success
  ##                       (second lvl)
  b[a] <- levels[2]

  ## otherwise not success (first lvl)
  b[!a] <- levels[1]

  ## we should return a factor
  return(factor(b, levels = levels))
}
preds.class <- toClass(preds,
                 levels(wines$quality))
```

5.5

```
## the function table builds contingency
## tables for the given factor variables
## be careful that they should have the
## same levels
tt <- table(preds.class, wines$quality)
tt
```

```
##
## preds.class bad good
##        bad  546  208
##        good 198  647
```

```
accuracy <- sum(diag(tt)) / sum(tt)
accuracy
```

```
## [1] 0.7460913
```

CORIS data

We load the data

```
coris <- read.table("coris.dat", skip = 4, sep = ",",
col.names = c("row.names", "sbp", "tobacco",
"ldl", "adiposity",
"famhist", "typea", "obesity",
"alcohol",
"age", "chd"))[,-1]
```

Ex 6

6.1

We use backward stepwise selection

```
res <- step(glm(chd ~ ., data = coris, family = "binomial"), direction =
              "backward", trace = 0)
summary(res)
```

```
##
## Call:
## glm(formula = chd ~ tobacco + ldl + famhist + typea + age, family = "binomial",
##     data = coris)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9165  -0.8054  -0.4430   0.9329   2.6139
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -6.44644    0.92087  -7.000 2.55e-12 ***
## tobacco      0.08038    0.02588   3.106  0.00190 **
## ldl          0.16199    0.05497   2.947  0.00321 **
## famhist      0.90818    0.22576   4.023 5.75e-05 ***
## typea        0.03712    0.01217   3.051  0.00228 **
## age          0.05046    0.01021   4.944 7.65e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 596.11  on 461  degrees of freedom
## Residual deviance: 475.69  on 456  degrees of freedom
## AIC: 487.69
##
## Number of Fisher Scoring iterations: 5
```

6.2

The complete model for logistic regression is

```
logreg_all <- glm(chd ~ ., data = coris, family = "binomial")
summary(logreg_all)
```

```
##
## Call:
## glm(formula = chd ~ ., family = "binomial", data = coris)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7781  -0.8213  -0.4387   0.8889   2.5435
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -6.1507209  1.3082600  -4.701 2.58e-06 ***
## sbp          0.0065040  0.0057304   1.135 0.256374
## tobacco      0.0793764  0.0266028   2.984 0.002847 **
## ldl          0.1739239  0.0596617   2.915 0.003555 **
## adiposity    0.0185866  0.0292894   0.635 0.525700
## famhist      0.9253704  0.2278940   4.061 4.90e-05 ***
## typea        0.0395950  0.0123202   3.214 0.001310 **
## obesity     -0.0629099  0.0442477  -1.422 0.155095
## alcohol      0.0001217  0.0044832   0.027 0.978350
## age          0.0452253  0.0121298   3.728 0.000193 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 596.11  on 461  degrees of freedom
## Residual deviance: 472.14  on 452  degrees of freedom
## AIC: 492.14
##
## Number of Fisher Scoring iterations: 5
```

We can see that the p-values for the coefficient of obesity and alcohol are not small, thus those variables seems to be less important in predicting coronary hear disease.