STK2100: Solutions Week 8

Lars H. B. Olsen

03.03.2021

Textbook

Exercise 5.4

The x-axis of a ROC-plot corresponds to the *false positive rate*

$$\text{FPR} = \text{fall-out} = 1 - \text{Specificity} = \frac{\text{False Positive}}{\text{False Positive} + \text{True Negative}}.$$

That is, the proportion of 'No' observations that is classified as calss 'Yes' by our classifier. The y-axis of a ROC-curve corresponds to the *true positive rate*

$$\text{TPR} = \text{Sensitivity} = \text{recall} = \text{hit-rate} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}.$$

That is, the proportion of 'Yes' observations that our classifier label as 'Yes'. If you are on the diagonal, this means that these two ratios are the same, which happens if the labels are chosen randomly.

Note that the book define the orientation of the x-axis and y-axis this way, but other books flip this, i.e., they have TPR on the x-axis and FPR on the y-axis.

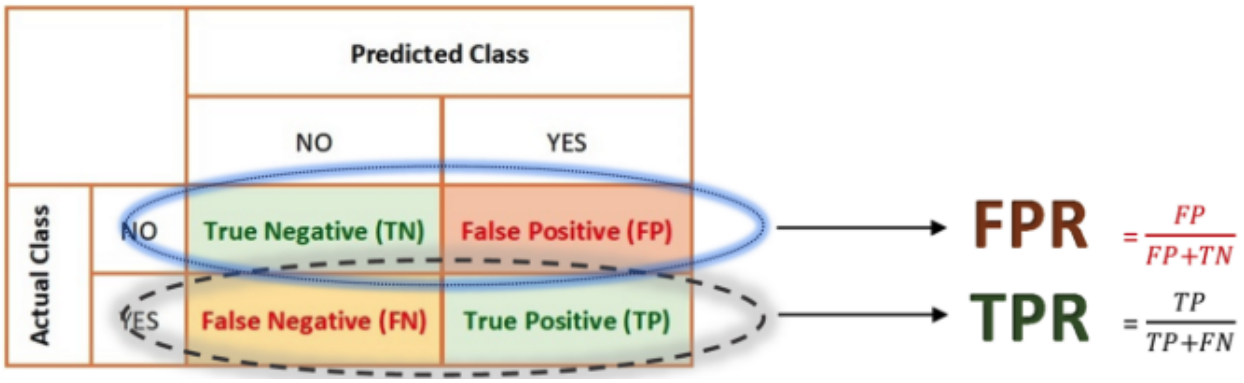


Figure 1: A figurative explanation of FPR and TPR.

To fully understand the diagonal line. First, think that we classify the labels at random, i.e., $\Pr(Yes) = \alpha$ and $\Pr(No) = 1 - \alpha$, where $\alpha \in [0, 1]$. Second, assume that our data consists of N_{No} observations from class 'No' and N_{Yes} observations from class 'Yes'. That is a total of $N = N_{No} + N_{Yes}$ observations. Then $\text{TN} = (1 - \alpha)N_{No}$, $\text{FP} = \alpha N_{No}$, $\text{FN} = (1 - \alpha)N_{Yes}$, and $\text{TP} = \alpha N_{Yes}$. Note that these sum to N. Furthermore, by simple algebra, we get that $\text{FPR} = \text{TPR} = \alpha$. Hence, we get the diagonal line by varying α from 0 to 1.

Especially, note that classifying with the help of a coin will yield the coordinate $(0.5, 0.5)$ in ROC-space.

For some nice explanatory figures, see https://dzone.com/articles/what-it-the-interpretation-of-the-diagonal-for-a-r.

Exercise 5.5

A binary classifier with a ROC curve that is completely under the diagonal is not necessarily a poor classifier. This behavior only means that the classifier label the observations in a flipped manner. That is, in stead of labeling observations with label 'Yes', it labels them with 'No'. This is easily fixed by flipping the labels, and voila, you got a good classifier.

You get a ROC curve that is partly under the diagonal when for example the standard deviations of the two outcomes are rather different. Look at the distribution plots in the first link below and and think what would happen if they did not have the same spread.

If you still not find ROC-curves intuitive, then you might want to read/look at the figures in https://towardsdatascience.com/demystifying-roc-curves-df809474529a and http://mlwiki.org/index.php/ROC_Analysis.

Extra exercise:

1. Simulate $X_1, \ldots, X_{40} \sim \mathcal{N}(0, 1^2)$ and $Y_1, \ldots, Y_{40} \sim \mathcal{N}(1, 3^2)$. Let the Xs have label 'No' and the Ys label 'Yes'.
2. Let the threshold value ω vary from -10 to 10.
3. Label everything left of ω as 'No' and everything right of ω as 'Yes'.
4. Plot the ROC. What do you see? How does the curve behave?

Exercise 5.7

Let $\sum_{k=0}^{K-1} \pi_k(x) = 1$, then $\sum_{k=1}^{K-1} \pi_k(x) = 1 - \pi_0(x)$. Assume that $\log \frac{\pi_k(x)}{\pi_0(x)} = \eta_k(x)$. Take the exponential on both sides

$$\frac{\pi_k(x)}{\pi_0(x)} = \exp\{\eta_k(x)\}. \tag{1}$$

Then take the sums from $k = 1$ to $K - 1$ on both sides

$$\begin{aligned}
\sum_{k=1}^{K-1} \frac{\pi_k(x)}{\pi_0(x)} &= \sum_{k=1}^{K-1} \exp\{\eta_k(x)\} \\
\frac{1}{\pi_0(x)} \sum_{k=1}^{K-1} \pi_k(x) &= \sum_{k=1}^{K-1} \exp\{\eta_k(x)\} \\
\frac{1 - \pi_0(x)}{\pi_0(x)} &= \sum_{k=1}^{K-1} \exp\{\eta_k(x)\} \\
\frac{1}{\pi_0(x)} - 1 &= \sum_{k=1}^{K-1} \exp\{\eta_k(x)\} \\
\frac{1}{\pi_0(x)} &= 1 + \sum_{k=1}^{K-1} \exp\{\eta_k(x)\} \\
\pi_0(x) &= \frac{1}{1 + \sum_{k=1}^{K-1} \exp\{\eta_k(x)\}}.
\end{aligned}$$

Furthermore, from equation (1), we have that

$$\begin{aligned}
\frac{\pi_k(x)}{\pi_0(x)} &= \exp\{\eta_k(x)\} \\
\pi_k(x) &= \pi_0(x) \exp\{\eta_k(x)\} \\
\pi_k(x) &= \frac{\exp\{\eta_k(x)\}}{1 + \sum_{j=1}^{K-1} \exp\{\eta_j(x)\}}
\end{aligned}$$

In the exercise text it states that $\eta_r(x_i) = \beta_0 + \sum_{j=1}^{p} x_{ij}\beta_{jr}$. And that we should use that (5.2) also holds with $r = 0$, by setting $\beta_{j0} = 0$, for $j = 0, 1, \ldots, p$. If (5.2) holds for $r = 0$, we get that

$$\log \frac{\pi_0(x)}{\pi_0(x)} = \log(1) = 0 \quad \Longrightarrow \quad \eta_k(x) = 0.$$

Which is the case as $\beta_{j0} = 0$, for $j = 0, 1, \ldots, p$. Furthermore, we then see that

$$\pi_0(x) = \frac{\exp\{\eta_0(x)\}}{1 + \sum_{j=1}^{K-1} \exp\{\eta_j(x)\}} = \frac{1}{1 + \sum_{j=1}^{K-1} \exp\{\eta_j(x)\}},$$

which is what we derived initially.

ISLR

Exercise 7.5 CONCEPTUAL: Smoothing Splines

Consider the two curves, $\hat{g}_1$ and $\hat{g}_2$, defined by

$$\hat{g}_1 = \arg\min_g \left(\sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \int [g^{(3)}(x)]^2 dx \right)$$

$$\hat{g}_2 = \arg\min_g \left(\sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \int [g^{(4)}(x)]^2 dx \right)$$

where $g^{(m)}$ represents the mth derivative of g.

a) Training RSS as $\lambda \to \infty$

Q: *As $\lambda \to \infty$, will $\hat{g}_1$ or $\hat{g}_2$ have the smaller training RSS?*

A:

As λ increases, the penalty term becomes more and more important in the above equation.

For $\hat{g}_1$, $\lambda \to \infty$ forces $g^{(3)}(x) \to 0$. The highest order (most flexible) polynomial to satisfy this would be of the form $\hat{g}(x) = ax^2 + bx + c$ (as the third derivative would be zero). $\hat{g}_1$ will therefore be the *quadratic* that minimizes the training RSS.

For $\hat{g}_2$, $\lambda \to \infty$ forces $g^{(4)}(x) \to 0$. The highest order (most flexible) polynomial to satisfy this would be of the form $\hat{g}(x) = ax^3 + bx^2 + cx + d$ (as the third derivative would be zero). $\hat{g}_2$ will therefore be the *cubic* that minimizes the training RSS.

Since $\hat{g}_2$ would be more flexible as the higher order polynomial, $\hat{g}_2$ would have the smaller training RSS.

b) Test RSS as $\lambda \to \infty$

Q: *As $\lambda \to \infty$, will $\hat{g}_1$ or $\hat{g}_2$ have the smaller test RSS?*

A:

We don't know - it depends whether the true relationship between x and y is better approximated with a cubic or quadratic relationship. It is possible that $\hat{g}_2$ could be overfit and have a larger test RSS, or $\hat{g}_1$ could be underfit and have a larger test RSS.

c) Training & Test RSS for $\lambda = 0$

Q: *For $\lambda = 0$, will $\hat{g}_1$ or $\hat{g}_2$ have the smaller training and test RSS?*

A:

For $\lambda = 0$ there is no restriction at all on $g(x)$, so both $\hat{g}_1$ and $\hat{g}_2$ would have the same training RSS (of zero if all the x_i are unique). With no restrictions on g we can simply have any function that interpolates all the training observations.

Both of these functions would be terribly overfit and have high test RSS. If we assume that the same interpolating function was chosen for $\hat{g}_1$ and $\hat{g}_2$ (e.g. both were a linear spline with knots at each unique x_i) they would also clearly have the same test RSS.

Exercise 7.11 APPLIED: Generated Data (GAM Backfitting, $p = 2$)

Disclaimer

The following solutions were provided by *Liam Morgan*. He uses the the `tidyverse` library which is a common package used in R to make data science faster, easier and more fun. The notation is a bit different from base R, which you are used to, but you should able to understand the new notation. See https://www.tidyverse.org for more on `tidyverse`. Especially important is the `%>%` notation, will simply means that it pass the left hand side of the operator to the first argument of the right hand side of the operator. Thus, e.g., `boston %>% head()` is equivalent to `head(boston)`.

In Section 7.7, it was mentioned that GAMs are generally fit using a *backfitting* approach. The idea behind backfitting is actually quite simple. We will now explore backfitting in the context of multiple linear regression.

Suppose that we would like to perform multiple linear regression, but we do not have software to do so. Instead, we only have software to perform simple linear regression. Therefore, we take the following iterative approach: we repeatedly hold all but one coefficient estimate fixed at its current value, and update only that coefficient estimate using a simple linear regression. The process is continued until *convergence* - that is, until the coefficient estimates stop changing.

We now try this out on a toy example.

a) Generate Data

Q: *Generate a response Y and two predictors X_1 and X_2, with $n = 100$.*

A:

I generate Y using the following model:

$$\begin{aligned} Y &= 16 + 5.1X_1 - 7.3X_2 + \epsilon \\ X_1 &\sim \mathcal{N}(0, 1) \\ X_2 &\sim \mathcal{N}(0, 1) \\ \epsilon &\sim \mathcal{N}(0, 1) \end{aligned}$$

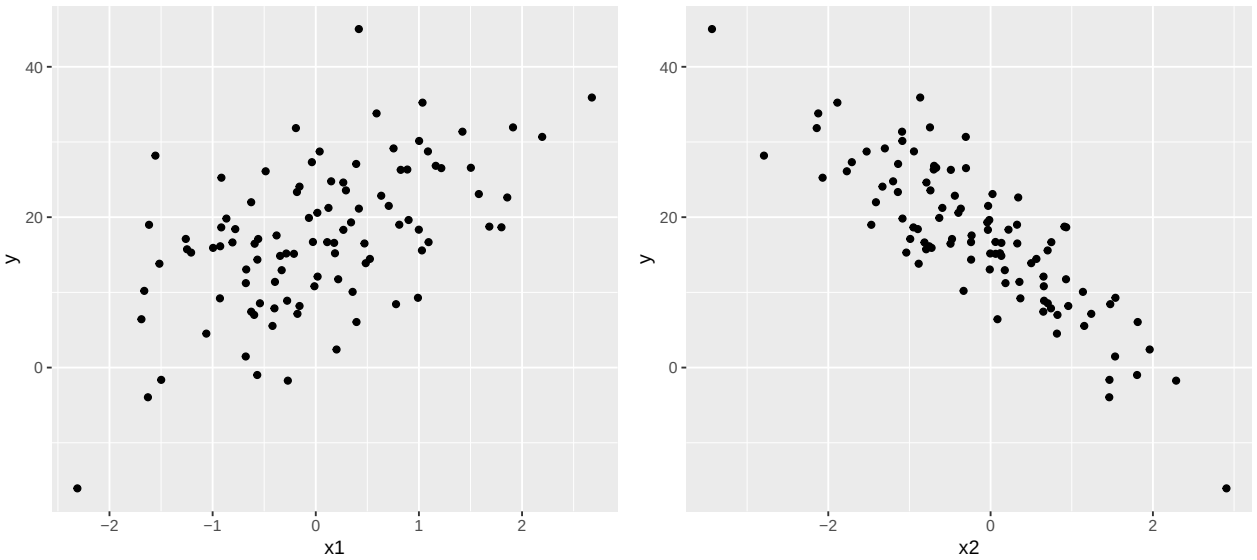
```
# The tidyverse is an opinionated collection of R packages
# designed for data science
library(tidyverse)
library(gridExtra) # combining graphs
select <- dplyr::select
```

```
set.seed(591)
x1 <- rnorm(100)
```

```
x2 <- rnorm(100)
eps <- rnorm(100)

y <- 16 + 5.1*x1 - 7.3*x2 + eps
```

Here are the relationships visualized across two dimensions:

```
g1 <- data.frame(x1, x2, y) %>%
  ggplot(aes(x1, y)) +
  geom_point()

g2 <- data.frame(x1, x2, y) %>%
  ggplot(aes(x2, y)) +
  geom_point()

grid.arrange(g1, g2, ncol = 2)
```



b) Initialize $\hat{\beta}_1$

Q: *Initialize $\hat{\beta}_1$ to take on a value of your choice. It does not matter what value you choose.*

A:

I will initialize $\hat{\beta}_1$ at 1:

```
beta_1 <- 1
beta_1
```

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

c) Simple Linear Regression - Estimating β_2

Q: *Keeping $\hat{\beta}_1$ fixed, fit the model*

$$Y - \hat{\beta}_1 X_1 = \beta_0 + \beta_2 X_2 + \epsilon$$

You can do this as follows:

```
r2 = y - beta_1 * x1
beta_2 = lm(r2 ~ x2)$coef[2]
```

A:

I change notation for the code a bit here.

Recall a couple of things from section 7.7 as it is useful to keep perspective on where this is used:

> Fitting a GAM with a smoothing spline is not quite as simple as fitting a GAM with a natural spline, since in the case of smoothing splines, least squares cannot be used. However, standard software such as the `gam()` function in `R` can be used to fit GAMs using smoothing splines, via an approach known as *backfitting*. This method fits a model involving multiple predictors by repeatedly updating the fit for each predictor in turn, holding the others fixed. The beauty of this approach is that each time we update a function, we simply apply the fitting method for that variable to a *partial residual*.

Therefore, we first calculated the partial residual for `x2` ('`r2`'), which is `y` with the contribution of all other explanatory variables subtracted (in this case, just `x1`): $r_2 = Y - \hat{\beta}_1 X_1$:

```
r2 <- y - beta_1 * x1
```

I then fit a simple linear regression *predicting* this partial residual using `x2`. Extracting the coefficient gives the initial value for $\hat{\beta}_2$:

```
beta_2 <- as.numeric(lm(r2 ~ x2)$coef[2])
beta_2
```

```
## [1] -7.396905
```

d) Simple Linear Regression - Estimating β_1

Q: *Keeping $\hat{\beta}_2$ fixed, fit the model*

$$Y - \hat{\beta}_2 X_2 = \beta_0 + \beta_1 X_1 + \epsilon$$

You can do this as follows:

```
r1 = y - beta_2 * x2
beta_1 = lm(r1 ~ x1)$coef[2]
```

A:

As before, but now for the partial residual for `x1` ('`r1`'). Note that we are now making use of the estimate $\hat{\beta}_2$ to replace the initial value of $\hat{\beta}_1$:

```
r1 <- y - beta_2 * x2

beta_1 <- as.numeric(lm(r1 ~ x1)$coef[2])
beta_1
```

```
## [1] 5.229975
```

Since we have fit the model $r_1 = \beta_0 + \beta_1 X_1 + \epsilon$, we can also get our estimate for β_0 by extracting the intercept coefficient:

```
beta_0 <- as.numeric(lm(r1 ~ x1)$coef[1])
beta_0
```

```
## [1] 16.13128
```

We are already amazingly close to the true estimates.

e) Looping (1,000 Iterations)

Q: *Write a for loop to repeat (c) and (d) 1,000 times. Report the estimates of $\hat{\beta}_0$, $\hat{\beta}_1$, and $\hat{\beta}_2$ at each iteration of the for loop. Create a plot in which each of these values is displayed, with $\hat{\beta}_0$, $\hat{\beta}_1$, and $\hat{\beta}_2$ each shown in a different color.*

A:

I initialize $\hat{\beta}_1$ again at 1. I do not consider this initialization as part of the first iteration:

```
beta_1_init <- 1
beta_1_init
```

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

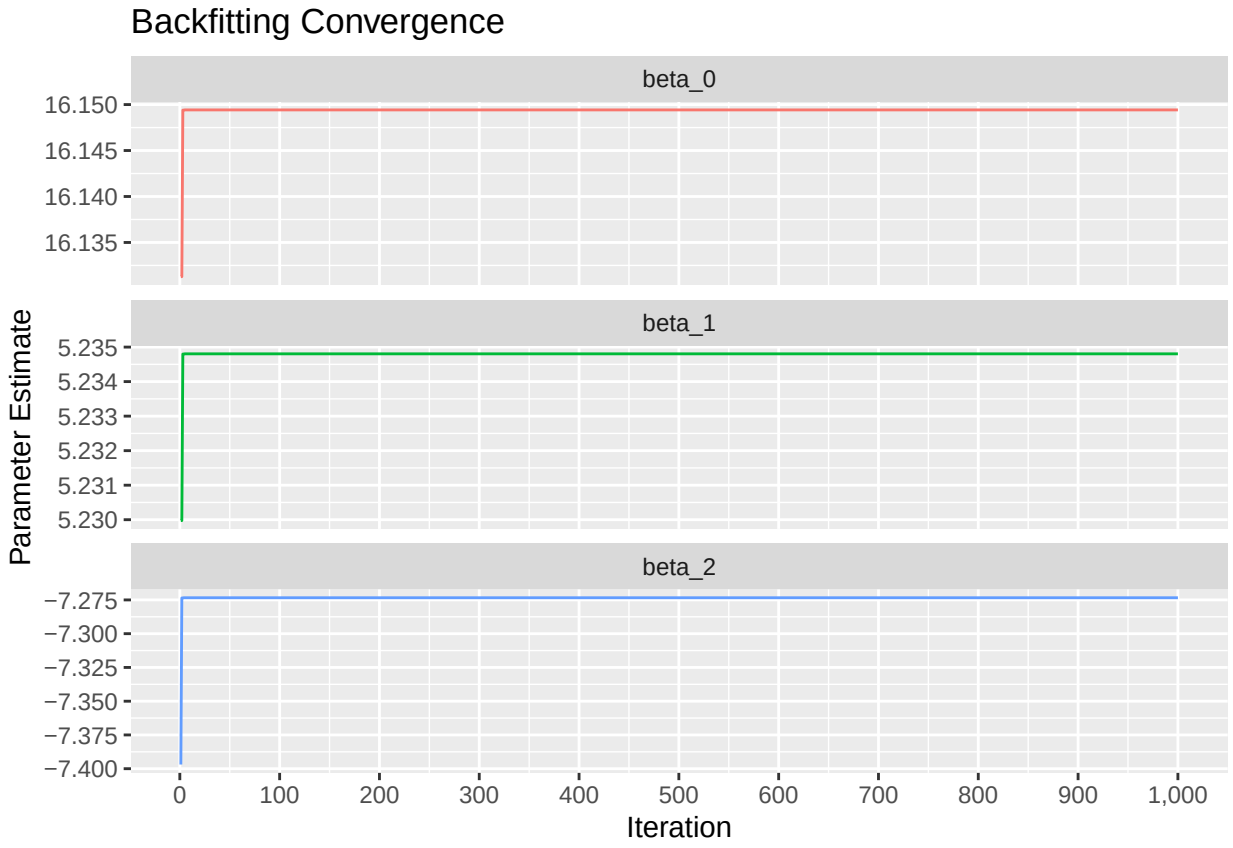
I perform the 1,000 iterations in the code below.

```
beta_0 <- c()
beta_1 <- c()
beta_2 <- c()

# beta_2:
r2 <- y - beta_1_init * x1
beta_2[1] <- lm(r2 ~ x2)$coef[2]

# beta_1:
r1 <- y - beta_2[1] * x2
lm_coef <- lm(r1 ~ x1)$coef
beta_1[1] <- lm_coef[2]

# beta_0:
beta_0[1] <- lm_coef[1]


for (i in 2:1000) {
  r2 <- y - beta_1[i-1] * x1
  beta_2[i] <- lm(r2 ~ x2)$coef[2]

  r1 <- y - beta_2[i-1] * x2
  lm_coef <- lm(r1 ~ x1)$coef
  beta_1[i] <- lm_coef[2]

  beta_0[i] <- lm_coef[1]
}
```

The question says that this process continues until *convergence*, where the estimated coefficients no longer change. It appears from the graph that this happens very quickly:

```
data.frame(iteration = 1:1000, beta_0, beta_1, beta_2) %>%
  pivot_longer(-iteration, names_to = "coefficient", values_to = "estimate") %>%
  ggplot(aes(x = iteration, y = estimate, col = factor(coefficient))) +
  geom_line() +
  scale_x_continuous(breaks = seq(0, 1000, 100), labels = scales::comma_format()) +
  facet_wrap(coefficient ~ ., scales = "free_y", nrow = 3) +
  theme(legend.position = "none") +
```

```
  labs(title = "Backfitting Convergence",
       x = "Iteration",
       y = "Parameter Estimate")
```



f) Comparison - Backfitting vs Multiple Regression

Q: *Compare your answer in (e) to the results of simply performing multiple linear regression to predict Y using X_1 and X_2. Use the* `abline()` *function to overlay those multiple linear regression coefficient estimates on the plot obtained in (e).*

A:

I simply fit the multiple regression $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon$ to the data.

The fitted coefficients:

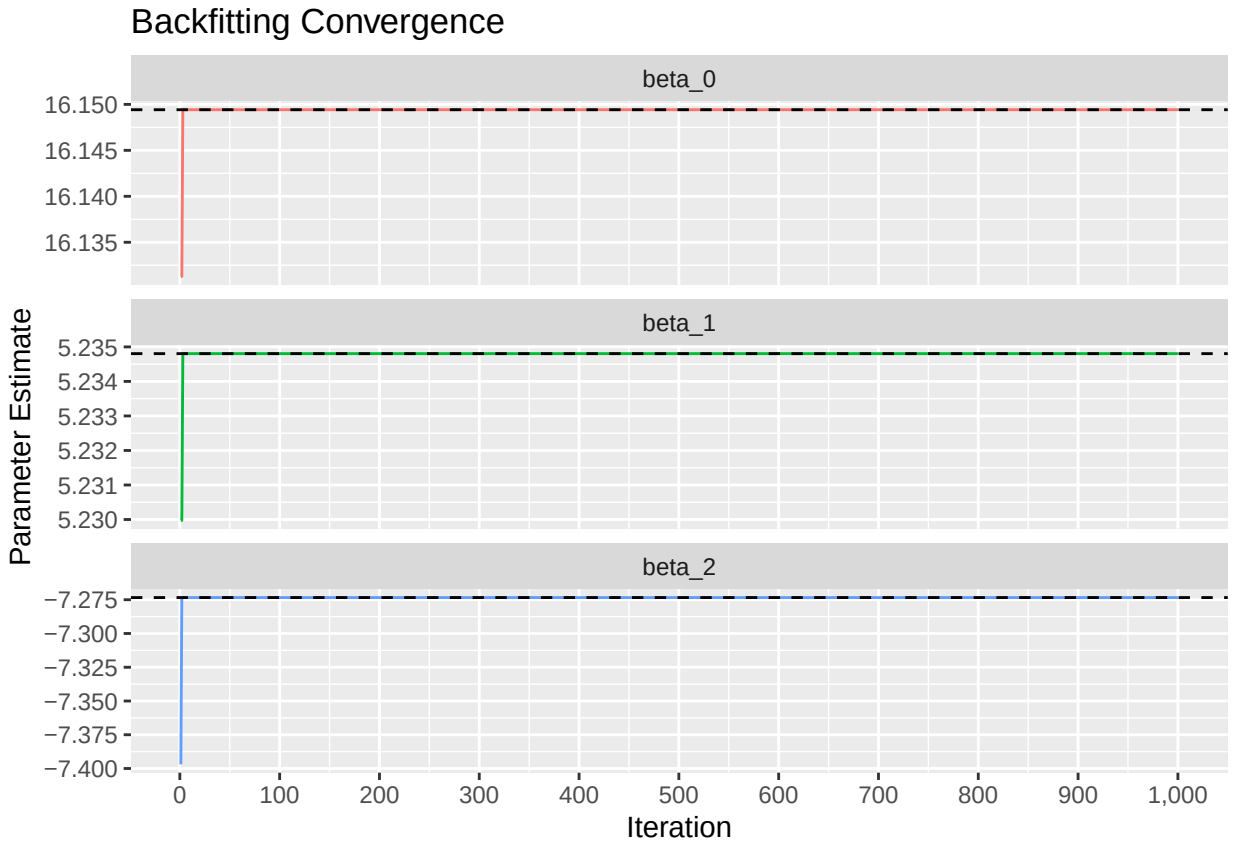
```
lm_coef <- lm(y ~ x1 + x2)$coefficients
lm_coef
```

```
## (Intercept)          x1          x2
##   16.149420    5.234804   -7.273397
```

Overlaying these coefficients onto the previous plot (with a dashed line), it appears that the looped simple linear regressions via backfitting gave the same (or very close) parameter estimates to those provided by multiple regression, once they had converged:

```
data.frame(iteration = 1:1000, beta_0, beta_1, beta_2) %>%
  pivot_longer(-iteration, names_to = "coefficient", values_to = "estimate") %>%
  ggplot(aes(x = iteration, y = estimate, col = factor(coefficient))) +
  geom_line() +
```

```
  geom_hline(data = data.frame(yint = lm_coef[1], coefficient = "beta_0"),
             aes(yintercept = yint), linetype = "dashed") +
  geom_hline(data = data.frame(yint = lm_coef[2], coefficient = "beta_1"),
             aes(yintercept = yint), linetype = "dashed") +
  geom_hline(data = data.frame(yint = lm_coef[3], coefficient = "beta_2"),
             aes(yintercept = yint), linetype = "dashed") +
  scale_x_continuous(breaks = seq(0, 1000, 100),
                     labels = scales::comma_format()) +
  facet_wrap(coefficient ~ ., scales = "free_y", nrow = 3) +
  theme(legend.position = "none") +
  labs(title = "Backfitting Convergence",
       x = "Iteration",
       y = "Parameter Estimate")
```



Extracting the backfitting coefficients at iteration 1,000 and comparing to those of multiple regression:

```
data.frame(simple_reg = c(beta_0[1000], beta_1[1000], beta_2[1000]),
           multiple_reg = as.numeric(lm_coef))
```

```
##   simple_reg multiple_reg
## 1  16.149420    16.149420
## 2   5.234804     5.234804
## 3  -7.273397    -7.273397
```

It certainly looks like they are identical, and the code below confirms.

```
identical(round(as.numeric(lm_coef), 12),
          round(c(beta_0[1000], beta_1[1000], beta_2[1000]), 12))
```

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

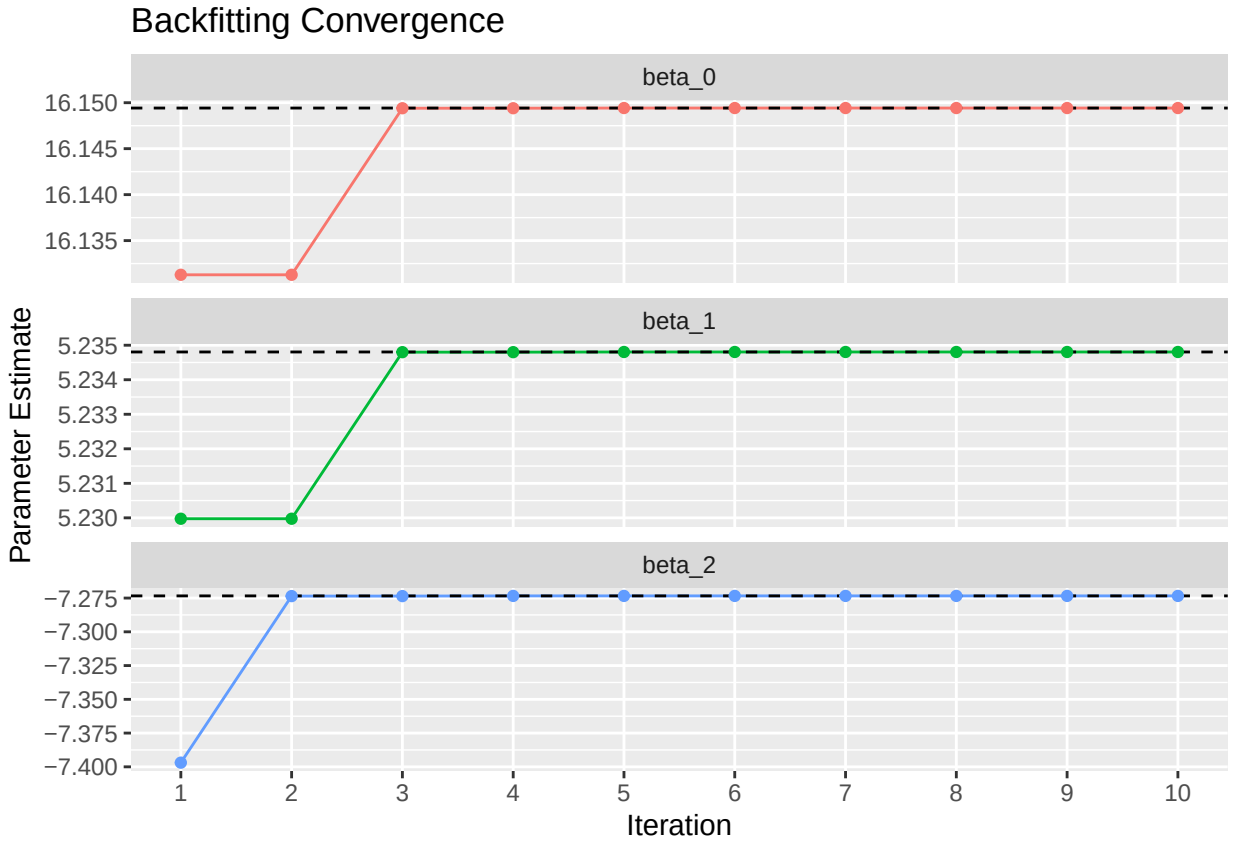
g) Backfitting - Convergence Iterations

Q: *On this data set, how many backfitting iterations were required in order to obtain a "good" approximation to the multiple regression coefficient estimates?*

A:

For this simple example, just one iteration gave us a 'good' approximation of multiple regression coefficients:

```
data.frame(iteration = 1:1000, beta_0, beta_1, beta_2) %>%
  pivot_longer(-iteration, names_to = "coefficient", values_to = "estimate") %>%
  ggplot(aes(x = iteration, y = estimate, col = factor(coefficient))) +
  geom_line() +
  geom_hline(data = data.frame(yint = lm_coef[1], coefficient = "beta_0"),
             aes(yintercept = yint), linetype = "dashed") +
  geom_hline(data = data.frame(yint = lm_coef[2], coefficient = "beta_1"),
             aes(yintercept = yint), linetype = "dashed") +
  geom_hline(data = data.frame(yint = lm_coef[3], coefficient = "beta_2"),
             aes(yintercept = yint), linetype = "dashed") +
  geom_point() +
  scale_x_continuous(limits = c(1, 10), breaks = seq(1, 10), minor_breaks = NULL) +
  facet_wrap(coefficient ~ ., scales = "free_y", nrow = 3) +
  theme(legend.position = "none") +
  labs(title = "Backfitting Convergence",
       x = "Iteration",
       y = "Parameter Estimate")
```



It looks like convergence happens at iteration 3 from the graph, and that all subsequent iterations give the same estimates.

A closer inspection shows that perhaps the coefficients haven't completely converged by this point:

```
length(unique(round(beta_0, 6)[3:1000])) == 1 &
length(unique(round(beta_1, 6)[3:1000])) == 1 &
length(unique(round(beta_2, 6)[3:1000])) == 1
```

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

Which iteration achieves 'convergence' happens seems to depend on our level of precision. For example, 6 decimal points (given by the multiple regression output) takes 5 iterations:

```
length(unique(round(beta_0, 6)[5:1000])) == 1 &
length(unique(round(beta_1, 6)[5:1000])) == 1 &
length(unique(round(beta_2, 6)[5:1000])) == 1
```

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

I presume that the number of iterations required for a 'good' approximation will vary significantly based on the strength of the relationships or number of variables.

Exercise 7.12 APPLIED: Generated Data (GAM Backfitting, $p = 100$)

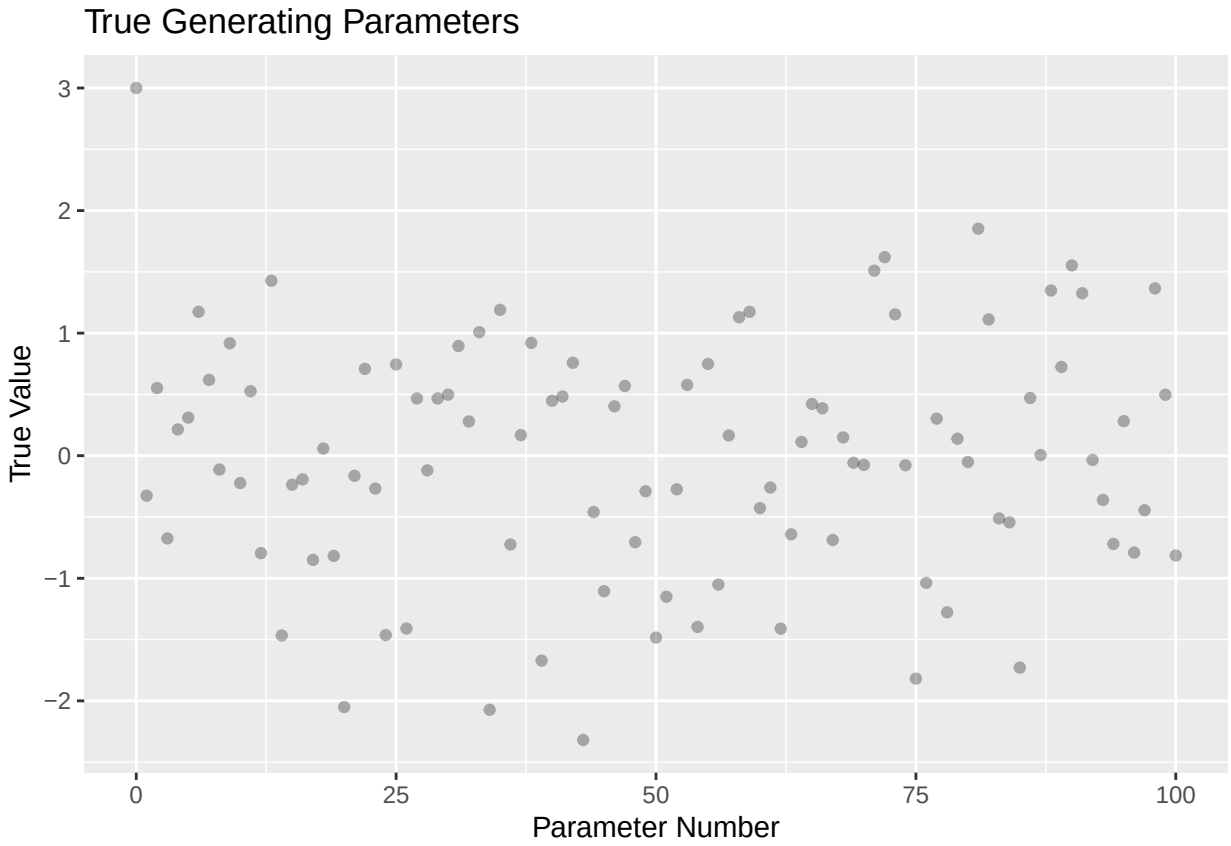
Q: *This problem is a continuation of the previous exercise. In a toy example with* $p = 100$*, show that one can approximate the multiple linear regression coefficient estimates by repeatedly performing simple linear regression in a backfitting procedure. How many backfitting iterations are required in order to obtain a "good" approximation to the multiple regression coefficient estimates? Create a plot to justify your answer.*

A:

I simulate the following relationship: $Y = \beta_0 + \sum_{i=1}^{100} \beta_i X_i + \epsilon$.

I first generate the 'true' parameters $\beta = (\beta_1, \beta_2, \ldots, \beta_{100})$. I generate the parameters from $\mathcal{N}(0, 1)$, with the exception of $\beta_0 =$ `beta_0`, which I arbitrarily set at 3:

```
set.seed(101)

betas <- rnorm(100)
beta_0 <- 3

ggplot(data.frame(parameter = 0:100, value = c(beta_0, betas)),
       aes(x = parameter, y = value)) +
  geom_point(alpha = 0.3) +
  labs(title = "True Generating Parameters",
       x = "Parameter Number",
       y = "True Value")
```



I generate $X = (X_1, X_2, \ldots, X_{100})$ from $\mathcal{N}(0, 1)$ again. I decide to go with a 1,000 observations, so X is a $1,000 \times 100$ matrix:

```
set.seed(102)
X <- matrix(rnorm(1000 * 100), nrow = 1000, ncol = 100)
dim(X)
```

```
## [1] 1000  100
```

I wanted to introduce significant noise so that the relationships aren't too strong. Here, $\epsilon \sim \mathcal{N}(0, 10^2)$:

```
set.seed(103)
noise <- rnorm(1000, sd = 10)
```

Finally I generate my response with $Y = \beta_0 + \sum_{i=1}^{100} \beta_i X_i + \epsilon$:

```
Y <- beta_0 + X %*% betas + noise
```

I fit a linear model and extracting the R^2 to this data to get an idea of the strength of the relationship between the response Y and the predictors in X:

```
summary(lm(Y ~ X))$r.squared
```

```
## [1] 0.528377
```

Backfitting:

I presume, based on the previous question, that the first step will need to be to provide an initial guess for all parameters except one. That is, we must initialize $\hat{\beta}_2, \hat{\beta}_2, \ldots, \hat{\beta}_{100}$. I take these as random estimates, as the previous question suggests that these initial values aren't important:

```
betas_iter <- rep(NA, 100)
beta_0_iter <- NA

set.seed(104)
betas_iter[2:100] <- rnorm(99)
```

I now try 50 iterations of the backfitting process, estimating the β_i one at a time in each iteration and updating the parameter estimates each time.

The first iteration will fit 100 models, updating the parameters one at a time:

1. $Y - \sum_{i=2}^{100} \hat{\beta}_i X_i = \beta_0 + \beta_1 X_1 + \epsilon$
2. $Y - \hat{\beta}_1 X_1 - \sum_{i=3}^{100} \hat{\beta}_i X_i = \beta_0 + \beta_2 X_2 + \epsilon$

...

99. $Y - \sum_{i=1}^{98} \hat{\beta}_i X_i - \hat{\beta}_{100} X_{100} = \beta_0 + \beta_{99} X_{99} + \epsilon$
100. $Y - \sum_{i=1}^{99} \hat{\beta}_i X_i = \beta_0 + \beta_{100} X_{100} + \epsilon$

On the fit for X_{100} I will also update the estimate for $\hat{\beta}_0$.

```
# rows = iteration (first = initial), cols = beta_0, beta_1, ..., beta_100
beta_matrix <- matrix(nrow = 1 + 50, ncol = 1 + 100)
beta_matrix[1, 3:101] <- betas_iter[2:100]

for (i in 1:50) {

  # update estimates for beta_1, ..., beta_100, then beta_0
  for (p in 1:100) {
    rp <- Y - X[ ,-p] %*% betas_iter[-p]
    betas_iter[p] <- lm(rp ~ X[ ,p])$coef[2]
  }

  beta_0_iter <- lm(rp ~ X[ ,p])$coef[1]

  # update matrix
  beta_matrix[i + 1, ] <- c(beta_0_iter, betas_iter)
}
```

I drop the first row of `beta_matrix`, since it only contains the initialized random values of $\hat{\beta}_2, \hat{\beta}_3, \ldots, \hat{\beta}_{100}$ that were required for the first simple linear regression.

```
beta_matrix <- beta_matrix[-1, ]
colnames(beta_matrix) <- c("(Intercept)", paste0("X", 1:100))
dim(beta_matrix)
```

```
## [1]  50 101
```

I display the first 20×6 entries of `beta_matrix`, which now takes the form:

$$\begin{bmatrix} \hat{\beta}_{0(1)} & \hat{\beta}_{1(1)} & \cdots & \hat{\beta}_{100(1)} \\ \hat{\beta}_{0(2)} & \ddots & & \\ \vdots & & \ddots & \\ \hat{\beta}_{0(50)} & & & \hat{\beta}_{100(50)} \end{bmatrix}$$

where $\hat{\beta}_{j(k)}$ is the estimate for β_j at iteration k.

```
beta_matrix[1:20, 1:6] %>% print(digits = 8)
```

```
##       (Intercept)          X1          X2         X3          X4          X5
##  [1,]   3.2392615 -1.70704102 -0.12680456 -1.2913319  0.49017672 -0.42537690
##  [2,]   3.2719487 -0.86775206  0.26510010 -1.1690149 -0.16608137  0.16336896
##  [3,]   3.2818133 -0.77487081  0.34143174 -1.2004922 -0.19236642  0.22145118
##  [4,]   3.2825939 -0.77934076  0.36537150 -1.2131186 -0.19448843  0.25010371
##  [5,]   3.2828479 -0.77968237  0.37034730 -1.2175314 -0.19330619  0.24948653
##  [6,]   3.2829583 -0.77969699  0.37061290 -1.2188563 -0.19344194  0.24843638
##  [7,]   3.2829878 -0.77961440  0.37049857 -1.2192055 -0.19353794  0.24815108
##  [8,]   3.2829950 -0.77957360  0.37044130 -1.2192901 -0.19356060  0.24808928
##  [9,]   3.2829968 -0.77955950  0.37042193 -1.2193107 -0.19356394  0.24807633
## [10,]   3.2829973 -0.77955533  0.37041627 -1.2193158 -0.19356425  0.24807372
## [11,]   3.2829974 -0.77955421  0.37041474 -1.2193170 -0.19356427  0.24807323
## [12,]   3.2829974 -0.77955392  0.37041434 -1.2193173 -0.19356427  0.24807315
## [13,]   3.2829974 -0.77955386  0.37041425 -1.2193173 -0.19356427  0.24807314
## [14,]   3.2829974 -0.77955384  0.37041422 -1.2193173 -0.19356428  0.24807314
## [15,]   3.2829974 -0.77955383  0.37041422 -1.2193173 -0.19356428  0.24807314
## [16,]   3.2829974 -0.77955383  0.37041421 -1.2193173 -0.19356428  0.24807314
## [17,]   3.2829974 -0.77955383  0.37041421 -1.2193173 -0.19356428  0.24807314
## [18,]   3.2829974 -0.77955383  0.37041421 -1.2193173 -0.19356428  0.24807314
## [19,]   3.2829974 -0.77955383  0.37041421 -1.2193173 -0.19356428  0.24807314
## [20,]   3.2829974 -0.77955383  0.37041421 -1.2193173 -0.19356428  0.24807314
```

I fit a multiple regression equation to the data and display the first 6 coefficients for a rough comparison.

```
lm_coef <- lm(Y ~ X)$coef
lm_coef[1:6] %>% print(digits = 8)
```

```
## (Intercept)          X1          X2          X3          X4          X5
##  3.28299744 -0.77955383  0.37041421 -1.21931735 -0.19356428  0.24807314
```

It looks like we have a "good" approximation of the multiple regression coefficients by the third iteration. While I am only viewing 6 of the 101 coefficients to be estimated, it seems likely that convergence happens to a high degree of precision somewhere around the 15th iteration. One potential (better) way of determining how 'close' these estimates are to the multiple regression estimates is shown below.

Parameter Distance:

To quantify how 'close' the parameters are to the multiple regression coefficients, I remembered the 'parameter distance' metric (see my solutions for chapter 6, question 10)g)). This seems reasonable to use again here; all the variables and corresponding parameter estimates are on the same-ish scale (the data was sampled from $\mathcal{N}(0, 1)$):

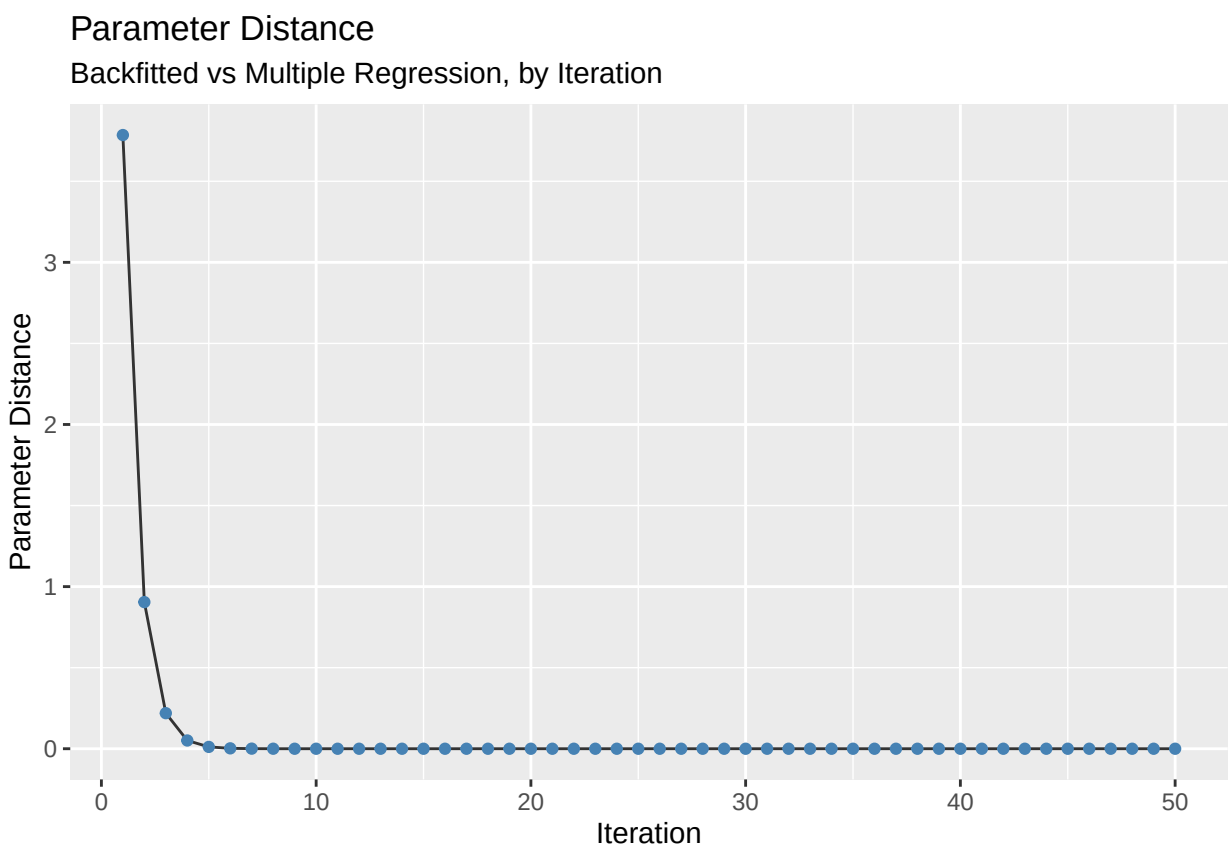
For each of the k iterations, I calculate the 'parameter distance' which I have denoted by d_k. I have modified the formula slightly from the chapter 6 solutions to:

- Include the intercept
- Calculate the distance between the backfitting coefficients $\beta_j\hat{(k)}$ and the multiple regression coefficients $\hat{\beta}_j^m$ (which do not change)

$$d_k = \sqrt{\sum_{j=0}^{p} \left(\hat{\beta}_{j(k)} - \hat{\beta}_j^m\right)^2}$$

I produce a plot of d_k by iteration:

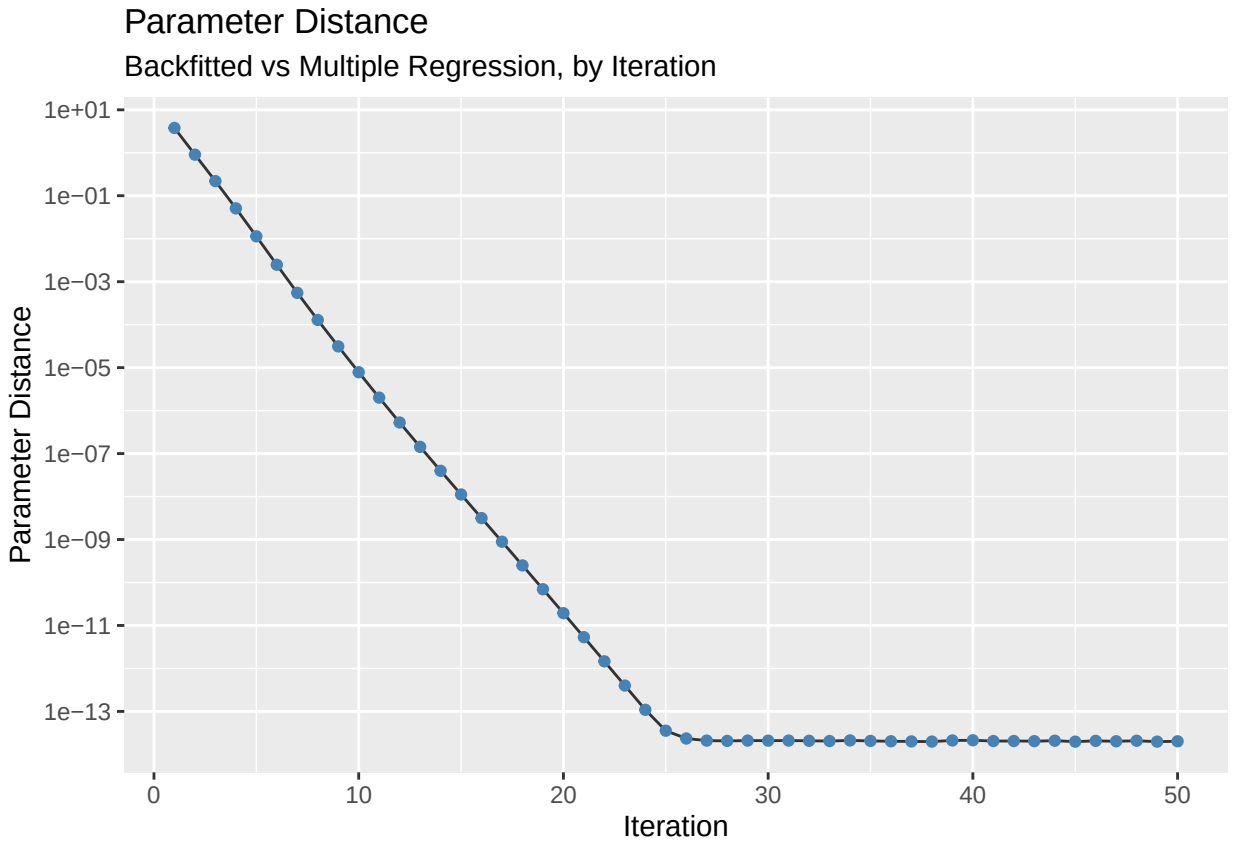
```r
param_dist <- c()

for (k in 1:50) {
  param_dist[k] <- sqrt(sum((beta_matrix[k, ] - lm_coef)^2))
}

data.frame(k = 1:50, param_dist = param_dist)  %>%
  ggplot(aes(x = k, y = param_dist)) +
  geom_line(col = "grey20") +
  geom_point(col = "steelblue") +
  scale_x_continuous(limits = c(1, 50), breaks = seq(0, 50, 10)) +
  labs(title = "Parameter Distance",
       subtitle = "Backfitted vs Multiple Regression, by Iteration",
       x = "Iteration",
       y = "Parameter Distance")
```



Perhaps a more useful way of displaying this graph is with the y axis on a log_{10} scale, where we can see the coefficients from the repeated simple linear regressions (backfitting) get *very* close to those given by multiple regression very quickly, and continue to converge to the multiple regression values with each iteration:

```r
data.frame(k = 1:50, param_dist = param_dist)  %>%
  ggplot(aes(x = k, y = param_dist)) +
  geom_line(col = "grey20") +
  geom_point(col = "steelblue") +
  scale_x_continuous(limits = c(1, 50), breaks = seq(0, 50, 10)) +
  scale_y_continuous(trans = "log10", breaks = 10^seq(1, -14, -2)) +
  labs(title = "Parameter Distance",
```

```
       subtitle = "Backfitted vs Multiple Regression, by Iteration",
       x = "Iteration",
       y = "Parameter Distance")
```



I presume the relatively flat line after iteration 25 is due to numeric limitations in R.

Exam STK2100 2017: Exercise 1

See https://www.uio.no/studier/emner/matnat/math/STK2100/oppgaver/STK2100_2017_fasit.pdf. It is in Norwegian, so if you you're not Norwegian, I would recommend to use Google Translate on the text, while the math is universal.