CompStat/R - Paper 3

Group 2: Carlo Michaelis, Patrick Molligo, Lukas Ruff

06 July 2016

Part I: Linear regression

In this first part of the paper, we will program a function which estimates the unknown parameters β and σ of a (ordinary) linear regression model

$$y = X\beta + \varepsilon, \qquad \varepsilon \sim N(0, \sigma^2 I)$$

by the ordinary least squares (OLS) method. For a given design matrix X and response vector y the OLS estimator is given by

$$\hat{\beta} = (X'X)^{-1}X'y \tag{1}$$

with covariance matrix

$$\mathrm{Var}(\hat{\beta}) = \sigma^2(X'X)^{-1} \tag{2}$$

where σ^2 has to be estimated via the sum of squared residuals (SSR):

$$\hat{\sigma}^2 = \frac{SSR}{df} = \frac{\sum_i (y_i - x_i'\hat{\beta})^2}{n-k}. \tag{3}$$

The term *df* refers to the degrees of freedom, i.e. the difference between the number of observations n and the number of coefficients k.

Raw implementation

The function `linModEst` is a raw implementation of the OLS estimator. The function takes the response vector y (`y`) and design matrix X (`x`) as arguments and returns a list with the following named elements:

- `coefficients`: the estimated coefficients $\hat{\beta}$
- `vcov`: the estimated covariance matrix $\widehat{\mathrm{Var}(\hat{\beta})}$
- `sigma`: the square root of the estimated scale parameter $\hat{\sigma}^2$
- `df`: the degrees of freedom *df*

We use equations (1), (2) and (3) for the implementation and compute the inverse of $X'X$ using the `solve` function, which numerically solves the equation

$$(X'X)\,A = I$$

for the matrix $A = (X'X)^{-1}$. To efficiently compute $X'X$ and $X'y$, we use the `crossprod` function.

```r
linModEst <- function(x, y) {
  # Computes the OLS estimator and sample variance assuming a (ordinary) linear
  # regression model.
  #
  # Args:
  #   x: design matrix x
  #   y: response vector y
  #
  # Returns:
  #   A list with the following named elements:
  #     $coefficients: the estimated coefficients
  #     $vcov: the estimated covariance matrix
  #     $sigma: the square root of the estimated variance
  #     $df: the degrees of freedom in the model, i.e. the difference between
  #          the number of rows and columns of x

  # Compute the inverse of (x'x) using the solve- and crossprod-function
  inv <- solve(crossprod(x), diag(nrow = ncol(x)))

  # Compute beta, i.e. the estimated coefficients
  coefficients <- inv %*% crossprod(x, y)

  # Compute the degrees of freedom
  df <- nrow(x) - ncol(x)

  # Compute the sample variance via the sum of squared residuals (SSR)
  SSR <- sum((y - x %*% coefficients)^2)
  sigma.squared <- SSR / df

  # Compute the covariance matrix
  vcov <- sigma.squared * inv

  # Create results list to be returned
  results <- list(coefficients, vcov, sqrt(sigma.squared), df)
  names(results) <- c("coefficients", "vcov", "sigma", "df")

  # Return results
  results
}
```

We test our implementation by computing the linear relationship between heart weight, body weight and sex for the `cats` dataset contained in the package `MASS`. In the following piece of code, `cbind` combines its arguments by columns into a matrix with the number of columns given by the number of arguments and the number of rows given by the greatest length of the given arguments. Shorter arguments are repeated, as long as the matrix number of rows is a multiple of the shorter vector lengths. Hence, `cbind(1, cats$Bwt, as.numeric(cats$Sex) - 1)` creates a design matrix with an intercept, the variable body weight (`bwt`) and sex (`Sex`), where the sex variable is converted from factor into a dummy variable. Therefore, `cbind` is used to build a proper design matrix of object type `matrix` with intercept and dummy variable, such that our implementation of `linModEst` works correctly.

```r
# Load cats dataset
data(cats, package = "MASS")

# Compute OLS using our implementation
linModEst(
  x = cbind(1, cats$Bwt, as.numeric(cats$Sex) - 1),
  y = cats$Hwt
)
```

```
## $coefficients
##             [,1]
## [1,] -0.41495263
## [2,]  4.07576892
## [3,] -0.08209684
##
## $vcov
##             [,1]        [,2]        [,3]
## [1,]  0.52900070 -0.20504763  0.06563743
## [2,] -0.20504763  0.08690026 -0.04696312
## [3,]  0.06563743 -0.04696312  0.09244480
##
## $sigma
## [1] 1.457138
##
## $df
## [1] 141
```

We verify our results by comparing the results to the output of `R`'s `lm` function:

```r
summary(lm(Hwt ~ Bwt + Sex, data = cats))
```

```
##
## Call:
## lm(formula = Hwt ~ Bwt + Sex, data = cats)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.5833 -0.9700 -0.0948  1.0432  5.1016
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -0.4149     0.7273  -0.571    0.569
## Bwt           4.0758     0.2948  13.826   <2e-16 ***
## SexM         -0.0821     0.3040  -0.270    0.788
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.457 on 141 degrees of freedom
## Multiple R-squared:  0.6468, Adjusted R-squared:  0.6418
## F-statistic: 129.1 on 2 and 141 DF,  p-value: < 2.2e-16
```

As we can see, our implementation is correct.

Extend implementation

In this section, we write a new function `linMod(formula, data)`, which estimates a linear regression model specified by `formula` and uses our `linModEst` function defined above to estimate the model parameters again by the OLS method. `linMod` returns a list with the following named elements:

- `coefficients`: named vector of the estimated coefficients $\hat{\beta}$
- `vcov`: named estimated covariance matrix $\widehat{\mathrm{Var}(\hat{\beta})}$
- `sigma`: the square root of the estimated scale parameter $\hat{\sigma}^2$
- `df`: the degrees of freedom df
- `formula`: the formula that represents the model equation
- `call`: the arguments with which `linMod` was called

Below, we use the `model.frame`, the `model.extract` and `model.matrix` functions which are very convenient for working with objects of the `formula` class. `model.frame` returns a `data.frame` containing only the variables from its passed `data` argument, which are used in the `formula` expression given. The returned `data.frame` from the `model.frame` function has also additional attributes, but those are not needed in our application. With `model.extract`, we are able to extract the response variable from the `data.frame` created by `model.frame`. Moreover, using `model.matrix`, we can create the design matrix (of object class `matrix`) again only from `formula` and `data` arguments. By default, the matrix returned by `model.matrix` includes an intercept and converts factor variables into proper dummy variables (i.e. a factor variable with L levels results in $L - 1$ dummy variables). More precisely, the default intercept is taken over from the `formula` object, which by default adds an intercept term to the model equation, if not specified otherwise. Finally, we use `match.call` to return the call of our function with all the specified arguments by their full names.

```r
linMod <- function(formula, data) {
  # Computes the OLS estimator and sample variance assuming a (ordinary) linear
  # regression model with model equation specified by the formula-argument.
  #
  # Args:
  #   formula: a formula specifying the linear model equation
  #   data: a data.frame, list or environment, containing the variables used in
  #         formula
  #
  # Returns:
  #   A list with the following named elements:
  #     $coefficients: named vector of the estimated coefficients
  #     $vcov: named estimated covariance matrix
  #     $sigma: the square root of the estimated variance
  #     $df: the degrees of freedom in the model
  #     $formula: the formula that represents the model equation
  #     $call: the arguments with which the function was called

  # Extract the response variable using the model.extract function on the
  # data.frame returned by model.frame
  y <- model.extract(model.frame(formula, data = data), "response")

  # Create the design matrix using model.matrix, which overtakes an intercept
  # specified in formula by default and converts factor variables into proper
  # dummy-variables
  x <- model.matrix(formula, data = data)

  # Use previously defined linModEst for estimation
  tmp <- linModEst(x, y)
```

```r
  # Prepare the output
  rownames(tmp$coefficients) <- colnames(x)
  colnames(tmp$vcov) <- colnames(x)
  rownames(tmp$vcov) <- colnames(x)

  # Create results list to be returned
  results <- c(tmp, formula, match.call())
  names(results) <- c("coefficients", "vcov", "sigma", "df", "formula", "call")

  # Return results
  results
}
```

Let's again test our implementation:

```r
linMod(Hwt ~ Bwt + Sex, data = cats)
```

```
## $coefficients
##                    [,1]
## (Intercept) -0.41495263
## Bwt          4.07576892
## SexM        -0.08209684
##
## $vcov
##             (Intercept)         Bwt        SexM
## (Intercept)  0.52900070 -0.20504763  0.06563743
## Bwt         -0.20504763  0.08690026 -0.04696312
## SexM         0.06563743 -0.04696312  0.09244480
##
## $sigma
## [1] 1.457138
##
## $df
## [1] 141
##
## $formula
## Hwt ~ Bwt + Sex
##
## $call
## linMod(formula = Hwt ~ Bwt + Sex, data = cats)
```

As we can see, the output has the desired format and the correct results.

Part II: S3 for linear models