LINEAR MODELS AND GLM: ASSIGNMENT 3 PARTIAL SAMPLE SOLUTIONS

Exercise 2: Implement an R function to perform Tukey's one degree of freedom test for non-additivity. Your function should accept a 2-dimensional table (matrix) as its argument. How would you use your function if the data is supplied as a data frame? [Hint: `?xtabs`]

Solution: Here is one implementation that returns a list containing the test statistic and the correponding p-value:

```
> tukey.1df <- function(y)
      # y: matrix of observations
  {
      df.num <- nrow(y) * ncol(y) - nrow(y) - ncol(y)
      y_.. <- mean(y)
      y_i. <- rowMeans(y)
      y_.j <- colMeans(y)
      C <- sum(y * ((y_i. - y_..) %*% t(y_.j - y_..)))
      D1 <- sum((y_i. - y_..)^2)
      D2 <- sum((y_.j - y_..)^2)
      MSI <- SSI <- C^2 / (D1 * D2)
      SSE <- sum((y - outer(y_i., y_.j, "+") + y_..)^2) - SSI
      MSE <- SSE / df.num
      pvalue <- pf(MSI/MSE, 1, df.num, lower.tail = FALSE)
      round(c(MSI = MSI, MSE = MSE, F = MSI/MSE, pvalue = pvalue), 4)
  }
```

Exercise 3: Use your function to perform the test for non-additivity in the `VADeaths` dataset.

Solution:

```
> tukey.1df(VADeaths)
    MSI     MSE       F  pvalue
68.9163  6.4057 10.7586  0.0073
```

Exercise 4: Implement an R function (or a suitable collection of R functions) to obtain an optimum Box-Cox transformation, including a profiled log-likelihood plot and a confidence interval.

Solution: It is best to implement a solution as a series of small functions. The main challenge is to manipulate the response in a formula-based implementation, because we need to fit a transformed model multiple times where the original response is replaced by a transformation. A simple solution is to coerce the formula into a character vector. The response will be the second component:

```
> as.character(rvar ~ a + b * c)
[1] "~"          "rvar"       "a + b * c"
```

We first define helper functions for transforming the data and computing the profiled log-likelihood.

```
> bctransform <- function(y, lambda)
  {
      if (lambda == 0) log(y)
      else (y^lambda - 1) / lambda
  }
> compute.loglik <- function(formula, data, lambda, sum.log.y)
  {
      fm <- lm(formula, data)
      RO2 <- sum(residuals(fm)^2)
      n <- nrow(data)
      Lmax <- -0.5 * n * (log(RO2/n) + log(2 * pi) + 1) + (lambda-1) * sum.log.y
  }
```

Next, we define the main function that computes the profiled log-likelihood values for a range of λ values.

```
> boxcox <- function(formula, data,
                     from = -2, to = 2, cuts = 101,
                     conf.level = 0.99)
  {
      resp.var <- as.character(formula)[2]
      y.values <- data[[ resp.var ]]
      if (any(y.values <= 0))
          stop("The response variable must be strictly positive")
      sum.log.y <- sum(log(y.values))
      lambda.grid <- seq(from = from, to = to, length = cuts)
      ploglik <- rep(NA, cuts)
      for (i in seq(along = lambda.grid))
      {
          data[[ resp.var ]] <-
              bctransform(y.values, lambda = lambda.grid[i])
          ploglik[i] <-
              compute.loglik(formula, data,
                             lambda = lambda.grid[i],
                             sum.log.y = sum.log.y)
      }
      ci.include <- (2 * (max(ploglik) - ploglik) < qchisq(conf.level, df = 1))
      ans <- list(lambda = lambda.grid, ploglik = ploglik,
                  conf.int = ci.include, conf.level = conf.level)
      class(ans) <- "lboxcox"
      ans
  }
```

This function returns an object of class “lboxcox”. We next want to define `plot()` and `print()` methods for such objects. The `print()` method will simply print the point estimate and the confidence interval.

```
> print.lboxcox <- function(x, ...)
  {
      cat(sprintf("Optimal Box-Cox parameter (%g confidence interval): ", x$conf.level))
      lambda.hat <- with(x, lambda[ which.max(ploglik) ])
      ci.start <- with(x, min(lambda[conf.int]))
```

```
      ci.end <- with(x, max(lambda[conf.int]))
      cat(sprintf("%g (%g, %g)\n", lambda.hat, ci.start, ci.end))
  }
```

Similarly, the `plot()` method will plot the profiled log-likelihood curve, and add vertical lines indicating the confidence interval.

```
> plot.lboxcox <- function(x, ...)
  {
      with(x,
       {
           plot(lambda, ploglik, type = "l",
                main = sprintf("Box-Cox transformation: %g confidence interval",
                               conf.level),
                xlab = expression(lambda),
                ylab = "Profiled log-likelihood")
           ci.start <- min(lambda[conf.int])
           ci.end <- max(lambda[conf.int])
           abline(v = c(ci.start, ci.end), col = "grey")
       })
  }
```

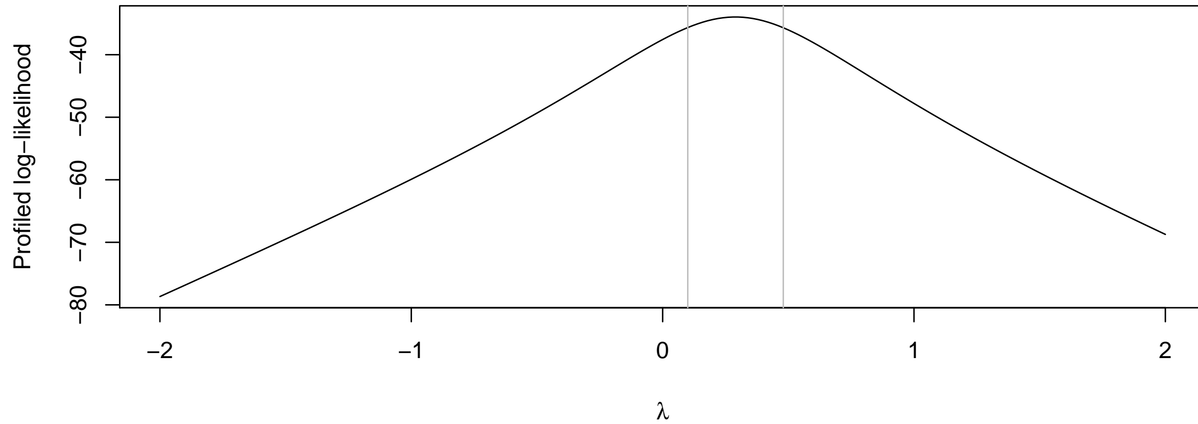
Exercise 5: Use your function to find an optimal power transformation in the `VADeaths` dataset. Plot the transformed values as before (see Assignment 2). Does the plot "look" more additive? Perform Tukey's test for non-additivity on the transformed data.

Solution: To use the formula interface, we first need to convert the `VADeaths` table into a data frame.

```
> names(dimnames(VADeaths)) <- c("AgeGroup", "PopGroup")
> VADeaths.df <- as.data.frame(as.table(VADeaths), responseName = "DeathRate")
> str(VADeaths.df)

'data.frame':        20 obs. of  3 variables:
 $ AgeGroup : Factor w/ 5 levels "50-54","55-59",..: 1 2 3 4 5 1 2 3 4 5 ...
 $ PopGroup : Factor w/ 4 levels "Rural Male","Rural Female",..: 1 1 1 1 1 2 2 2 2 2 ...
 $ DeathRate: num  11.7 18.1 26.9 41 66 8.7 11.7 20.3 30.9 54.3 ...
```

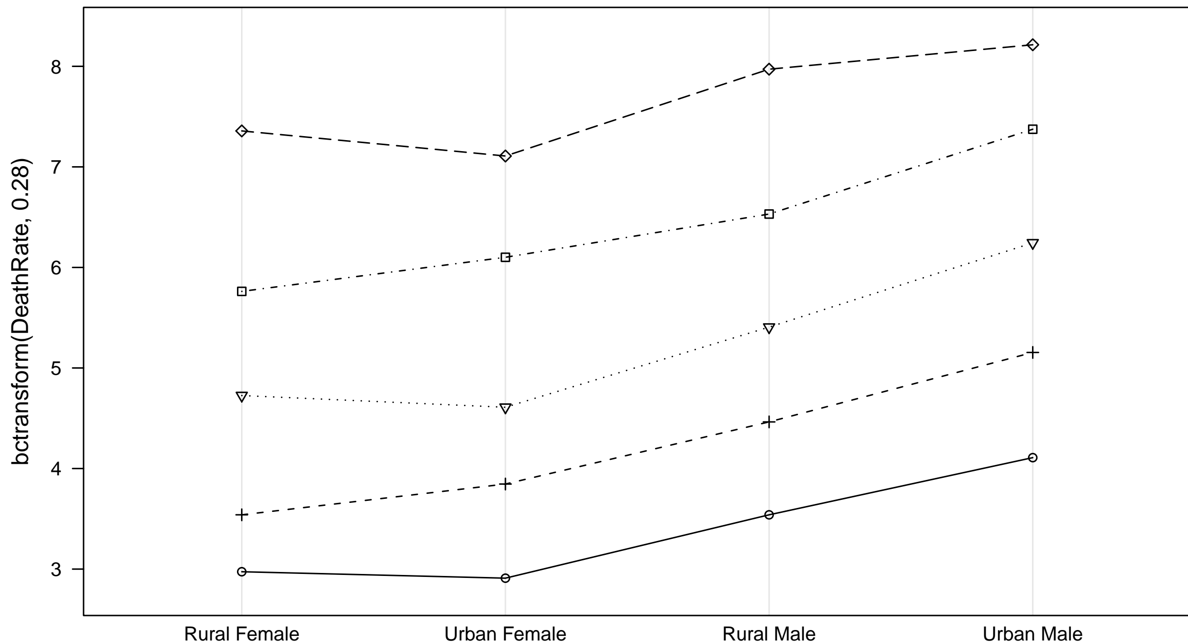
Next, we compute the profiled log-likelihoods by calling `boxcox()` and print and plot the result.

```
> bc.curve <- boxcox(DeathRate ~ AgeGroup + PopGroup, data = VADeaths.df,
                     conf.level = 0.95, cuts = 201)
> bc.curve

Optimal Box-Cox parameter (0.95 confidence interval): 0.28 (0.1, 0.48)

> plot(bc.curve)
```



The optimal λ is 0.28. The interaction plot of the transformed response is more consistent with additivity:

```
> library(lattice)
> dotplot(bctransform(DeathRate, 0.28) ~ reorder(PopGroup, DeathRate), data = VADeaths.df,
         groups = AgeGroup, type = c("p", "a"), auto.key = list(columns = 5))
```



Applying Tukey's test of non-additivity to the transformed data gives

```
> tukey.1df(bctransform(VADeaths, 0.28))
   MSI    MSE      F pvalue
0.0128 0.0293 0.4391 0.5212
```

Exercise 6: Write an R function to fit a linear model minimising the sum of absolute errors, using the technique of iteratively reweighted least squares (IRLS). You can use the `lm()` function with weights.

You will need to choose a convergence criterion. What criterion did you choose?

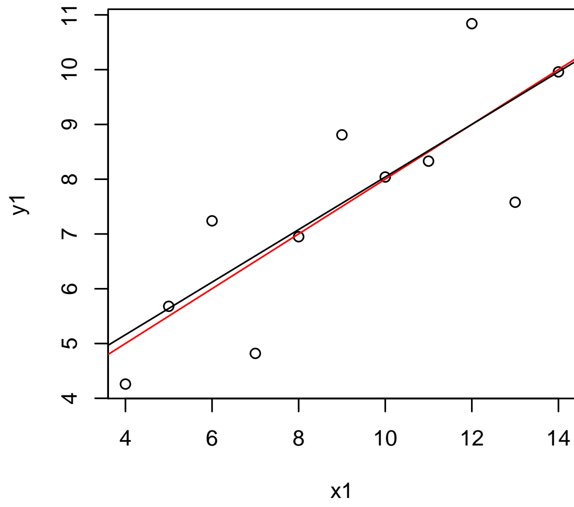
Solution: Here is one implementation. Note that it is not clear what one should do for observations where the current residual is zero. The original paper of Schlossmacher (`http://www.jstor.org/stable/2284512`) suggests setting the corresponding weights to zero (i.e., ignore these observations for the next least squares fit), but the justification for doing so is weak. In fact, doing so causes problems for our artificial datasets, and I have instead chosen to put an upper bound on the weights.

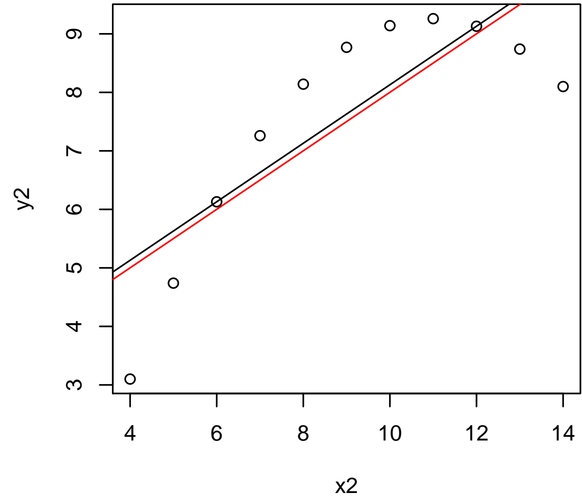
```
> ladlm <- function(formula, data, conv.cutoff = 1e-10, max.it = 200,
                    eps = 1e-10, plot = FALSE)
  {
      fm <- lm(formula, data)
      beta.ols <- beta.old <- coef(fm)
      for (i in seq_len(max.it))
      {
          r <- abs(resid(fm))
          data$lad.weights <- ifelse(r > eps, 1 / r, 1/eps)
          fm <- lm(formula, data, weights = lad.weights)
          beta.new <- coef(fm)
          if (sum((beta.new - beta.old)^2) / sum(beta.old^2) < conv.cutoff)
              break
          beta.old <- beta.new
      }
      if (plot)
      {
          plot(formula, data)
          abline(beta.ols, col = "red")
          abline(beta.new, col = "black")
      }
  }
```

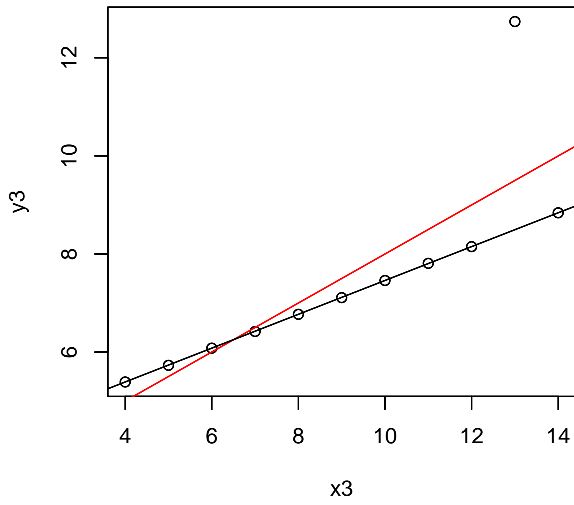
Exercise 7: Use the function written in the previous question to fit regression lines to Anscombe's four regression datasets (`?anscombe` in R). Produce a plot similar to what you get from `example(anscombe)`, but with regression lines fitted using your code instead of the usual least squares regression lines.

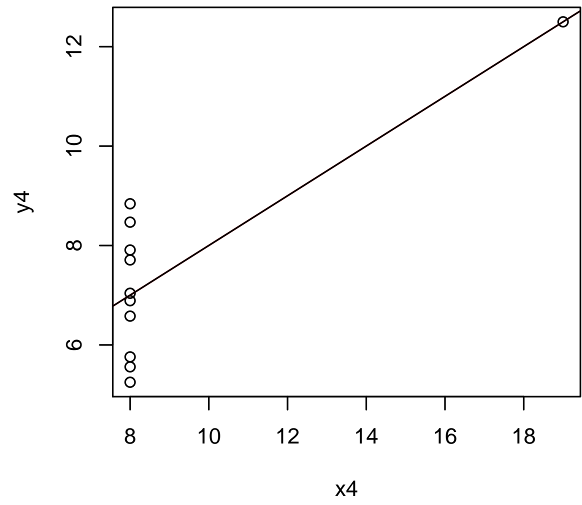
Solution.

```
> par(mfrow = c(2, 2))
> ladlm(y1 ~ x1, data = anscombe, plot = TRUE)
> ladlm(y2 ~ x2, data = anscombe, plot = TRUE)
> ladlm(y3 ~ x3, data = anscombe, plot = TRUE)
> ladlm(y4 ~ x4, data = anscombe, plot = TRUE)
```


y1
x1
4
5
6
7
8
9
10
11
4
6
8
10
12
14


y2
x2
3
4
5
6
7
8
9
4
6
8
10
12
14


y3
x3
6
8
10
12
4
6
8
10
12
14


y4
x4
6
8
10
12
8
10
12
14
16
18