```
###############################################################################################
# DESCRIPTION : the following are the R codes used for the numerical part of the
article. Some R packages  #
#                have been used to achieve the programme. To check the results we
got in our article, one   #
#                need to install and load those package in the R software. Then the
code should run succes- #
#                fully. Therefore, inside the code, when a package is load it's
needed to install that      #
#                package on your R
software.                                                        #
###############################################################################################
# The data
rm(list = ls())

#
--------------------------------------------------------------------------------------
# Auxiliary function: Generating the matrix of explanatory variables
# n is the number of coefficients
#
--------------------------------------------------------------------------------------

k <- 5

Xmat <- function(n){
  # Matrix that will merge all the Y and positions of i and j of th
  # UTM
  M <- matrix(0, nrow = n, ncol = n)


  # Code that select the indices of the upper triangular (UTM) matrix
  for (i in 1:n) {
    for (j in 1:n) {
      if (i + j <= (n+1)){
        M[i,j] <- paste("Y", i,j, sep = "_")
      } else {
        break
      }
    }
  }

  # Vector of indices
  # We will use them to extract the positions of i and j
  A <- as.character(t(M))
  A <-  A[A!="0"]

  # Use A to create the matrix X

  n_col <- length(c(1, rep(0,2*(n-1))))

  pos_i <- c(1, seq(2, n_col, by = 2))
  pos_j <- seq(1, n_col, by = 2)

  X <- matrix(0, nrow = length(A), ncol = n_col)
  X[,1] <- 1
  rownames(X) <- A # Can be omitted

  for (k in 1:length(A)) {
    i <- as.numeric(strsplit(A[k], split = "_")[[1]][2])
    j <- as.numeric(strsplit(A[k], split = "_")[[1]][3])

    X[k, ][pos_i[i]] <- 1
    X[k, ][pos_j[j]] <- 1
  }

  #
```

```r
  return(X)

}

# ------------------------------------------------------------------------------
#                          Classical Log-Poisson model                         #
# ------------------------------------------------------------------------------
k = 5
n <- k*(k+1)/2
Triangle <- data.frame(originf = as.factor(rep(1:k, k:1)),
                       devf=as.factor(sequence(k:1)),
                       inc.paid=
c(1120,2090,2610,2920,3130,1030,1920,2370,2710,1090,2140,2610,1300,2650,1420))

# The Poisson model

RegPoisson <- glm(inc.paid ~ originf + devf, data=Triangle, family=poisson(link =
"log"))
summary(RegPoisson)

# Prediction of the incremental claims payments

allClaims <- data.frame(origin = sort(rep(1:k, k)), dev = rep(1:k,k))
allClaims <- within(allClaims, {
  devf <- as.factor(dev)
  cal <- origin + dev - 1
  originf <- factor(origin)
})
(pred.inc.tri <- t(matrix(predict(RegPoisson,type="response", newdata=allClaims),
k, k)))

# The total Loss Reserve

sum(predict(RegPoisson,type="response", newdata=subset(allClaims, cal > k)))

# MSE from the package
#install.packages("Metrics")
library(Metrics)
pred.inc <- c(pred.inc.tri[1,], pred.inc.tri[2,1:4], pred.inc.tri[3, 1:3],
pred.inc.tri[4, 1:2], pred.inc.tri[5,1])
mse(Triangle$inc.paid, pred.inc)
# ------------------------------------------------------------------------------
#                               Overdispersion test                            #
# ------------------------------------------------------------------------------

library(AER)
if (!require(AER)) install.packages("AER")

dispersiontest(RegPoisson)

# ------------------------------------------------------------------------------

# ---------------------------------------------------------------- 8/9/2018
----------------------------
#
library(dplyr)
inc.paid1= c(1120,2090,2610,2920,3130,1030,1920,2370,2710, NA, 1090,2140,2610, NA,
NA, 1300,2650, NA, NA, NA,
            1420, NA, NA, NA, NA)

f <- function(k, values = inc.paid){ #
  # Matrix that will merge all the Y and positions of i and j of th
  # UTM
  M <- matrix(0, nrow = k, ncol = k)
  # Code that select the indices of the upper triangular (UTM) matrix
  for (i in 1:k) {
    for (j in 1:k) {
        M[i,j] <- paste("Y", i,j, sep = "_")
    }
  }

  A <- as.character(t(M))

  Df <- data.frame(idx = A, Y = inc.paid1) #
  return(Df)
}


Data <- f(5)

Data$nu <- log(as.numeric(t(pred.inc.tri)))

#
library("minqa")
fr <- function(nu) {
  (nu[2] - nu[1])^2 + (nu[4] - nu[3])^2 + (nu[6] - nu[5])^2 + (nu[8] - nu[7])^2 +
(nu[10] - nu[9])^2 + (nu[12] - nu[11])^2 + (nu[14] - nu[13])^2 + (nu[16] -
nu[15])^2 + (nu[18] - nu[17])^2 + (nu[20] - nu[19])^2 + (nu[22] - nu[21])^2 +
(nu[24] - nu[23])^2 + (nu[26] - nu[25])^2 + (nu[28] - nu[27])^2 + (nu[30] -
nu[29])^2
}
 h = seq(from = 0, to = 0.5, length = 101)
 h <- h[h > 0 ]
paramet <-matrix(NA, nrow = 30, ncol =100)
optival <- c()
N.iteration <- c()
library(dplyr)
Reserves <- c()
nu.est <- matrix(NA, nrow = 30, ncol =100)
MSEP.F <- c()
for (aa in 1:length(h)) {
  Data <- Data %>%
    mutate(logY = log(Y)) %>%
    mutate(test.data = (logY - h[aa]*nu)/(1-h[aa]))
  lower1 = c(-Inf, Data[1,5], -Inf, Data[2,5], -Inf, Data[3,5], -Inf, Data[4,5], -
Inf, Data[5,5], -Inf, Data[6,5], -Inf, Data[7,5], -Inf, Data[8,5], -Inf, Data[9,5],
-Inf, Data[11,5], -Inf, Data[12,5], -Inf, Data[13,5], -Inf, Data[16,5], -Inf,
Data[17,5], -Inf, Data[21,5])

  upper1 = c(Data[1,5], Inf, Data[2,5], Inf, Data[3,5], Inf, Data[4,5], Inf,
Data[5,5], Inf, Data[6,5], Inf, Data[7,5], Inf, Data[8,5], Inf, Data[9,5], Inf,
Data[11,5], Inf, Data[12,5], Inf, Data[13,5], Inf, Data[16,5], Inf, Data[17,5],
Inf, Data[21,5], Inf)
  nu0 <- c(Data[1,5]-4, Data[1,5] + 4, Data[2,5]-4, Data[2,5]+4, Data[3,5]-4,
Data[3,5]+4, Data[4,5]-4, Data[4,5]+4, Data[5,5]-4, Data[5,5]+4, Data[6,5]-4,
Data[6,5]+4, Data[7,5]-4, Data[7,5]+4, Data[8,5]-4, Data[8,5]+4, Data[9,5]-4,
Data[9,5]+4, Data[11,5]-4, Data[11,5]+4, Data[12,5]-4, Data[12,5]+4, Data[13,5]-4,
Data[13,5]+4, Data[16,5]-4, Data[16,5]+4, Data[17,5]-4, Data[17,5]+4, Data[21,5]-4,
Data[21,5]+4)
  res <- bobyqa(nu0, fr, lower = lower1, upper = upper1)
  paramet[,aa] <- res$par
  for (i in 1:30){
    if(i%%2 != 0){
      nu.est[i,aa] <-(paramet[i,aa] +  paramet[i+1,aa])/2
    }
  }
  Reserves[aa] <- exp(nu.est[11,aa] + nu.est[9,aa] - nu.est[1,aa]) +
exp(nu.est[19,aa] + nu.est[7,aa] - nu.est[1,aa]) + exp(nu.est[19,aa] + nu.est[9,aa]
- nu.est[1,aa]) + exp(nu.est[25,aa] + nu.est[5,aa] - nu.est[1,aa]) +
exp(nu.est[25,aa] + nu.est[7,aa] - nu.est[1,aa]) + exp(nu.est[25,aa] +
nu.est[9,aa]
- nu.est[1,aa]) + exp(nu.est[29,aa] + nu.est[3,aa] - nu.est[1,aa]) +
exp(nu.est[29,aa] + nu.est[5,aa] - nu.est[1,aa]) + exp(nu.est[29,aa] + nu.est[7,aa]
- nu.est[1,aa]) + exp(nu.est[29,aa] + nu.est[9,aa] - nu.est[1,aa])
  MSEP.F[aa] <- ((exp(nu.est[1,aa]) - Data$Y[1])^2 + (exp(nu.est[3,aa]) -
Data$Y[2])^2 + (exp(nu.est[5,aa]) - Data$Y[3])^2 + (exp(nu.est[7,aa]) -
Data$Y[4])^2 + (exp(nu.est[9,aa]) - Data$Y[5])^2 + (exp(nu.est[11,aa]) -
Data$Y[6])^2 + (exp(nu.est[13,aa]) - Data$Y[7])^2 + (exp(nu.est[15,aa]) -
Data$Y[8])^2 + (exp(nu.est[17,aa]) - Data$Y[9])^2 + (exp(nu.est[19,aa]) -
Data$Y[11])^2 + (exp(nu.est[21,aa]) - Data$Y[12])^2 + (exp(nu.est[23,aa]) -
Data$Y[13])^2 + (exp(nu.est[25,aa]) - Data$Y[16])^2 + (exp(nu.est[27,aa]) -
Data$Y[17])^2 + (exp(nu.est[29,aa]) - Data$Y[21])^2)/15
}
# Graphic of the evolution of Reserve, MSE with respect to h
library(ggplot2)
library(reshape)
Results <- as.data.frame(cbind(h, Reserves, MSEP.F))
Results1 <- melt(Results, id = 'MSEP.F')
Results <- as.data.frame(cbind(h, Reserves, MSEP.F))
Results1 <- melt(Results, id = 'h')
ggplot(Results1, aes(x = h, y = value, colour = variable)) +
  geom_line() +
  ylab(label="Evolution of Reserves and MSEP") +
  xlab("Values of h") +
  scale_colour_manual(values=c("red", "blue"))
# Values of Reserves and MSE for which h is Optimal
MSEP <- min(MSEP.F)
optimal.h <- h[which.min(MSEP.F)]
optimal.Reserves <- Reserves[which.min(MSEP.F)]
list(MSEP=MSEP, optimal.Reserves=optimal.Reserves, optimal.h = optimal.h)
```