# Algorismes IRWLS

[redacted]

May 2015

# 1 Introduction

This report studies the Iteratively Re-Weighted Least Squares (IRWLS) algorithm for finding the Maximum Likelihood (ML) estimators of a Generalised Linear Model (GLM) coefficients. The data is related to metal treatments. Four different temperatures $z_i, \quad i \in \{1, \cdots, 4\}$ are applied to metal ingots, and the result is evaluated with a binary variable which explains whether the ingot is ready for rolling or not. Let $x_i$ be the number of successes (treatments not ready) and $n_i$ the number of ignots tested in the conditions of temperature $z_i$

| $n_i$ trials | $x_i$ notready | $z_i$ heat |
|---|---|---|
| 55 | 0 | 7 |
| 157 | 2 | 14 |
| 159 | 7 | 27 |
| 16 | 3 | 57 |

Table 1: Data for the study

In the first part of the practice the maximum likelihood estimators are found assuming that our data follows a Binomial distribution.

$$X_i \sim B(n_i, p_i(\beta)), \quad i = 1, \cdots, 4, \quad \beta = (\beta_0, \beta_1), \quad p_i(\beta) = \frac{e^{\beta_0 + \beta_i \cdot z_i}}{1 + e^{\beta_0 + \beta_i \cdot z_i}}$$

For this purpose, the **optim** function of R is used. This function implements solvers for unconstrained optimization problems. With the transformation $k_{(x)} = x/(1+x)$ we have a problem without restrictions on the parameters. Different solvers are tested, including the derivative-free Nelder-Mead method.

In the second part of the report a different model is assumed. A GLM of the poisson family is considered with a logarithmic link function.

$$X_i \sim Poisson(\lambda_i), \quad i = 1, \cdots, 4$$
$$log(\lambda_i) = \beta_0 + \beta_1 \cdot n_i + \beta_2 \cdot z_i, \quad i = 1, \cdots, 4$$

The algorithm IRWLS is derived from the Fisher Scoring method and implemented in R. Its results are compared with the function *glm* to prove that the same result is attained.

# 2 Binomial Distribution Assumption

## 2.1 Likelihood Function

It is known that for one Binomial distribution the probability density function is

$$f_{x_i} = \binom{n_i}{x_i} \cdot p_i^{x_i} \cdot (1 - p_i)^{n_i - x_i}$$

The log-likelihood function of the four samples with the $p_i$ defined above altogether is

$$\mathcal{L} = \sum_{i=1}^{4} log \binom{n_i}{x_i} + x_i \cdot log(p_i) + (n_i - x_i) \cdot log(1 - p_i)$$

We can plot the contour of this function as it is 2-Dimensional with the following R-code

```
print.logll <-function(Beta0,Beta1,x1=0,x2=2,x3=7,x4=3,z1=7,z2=14,z3=27,z4=57,n1=55,n2=157,
                       n3=159,n4=16){
  log((factorial(n1)/(factorial(x1)*factorial(n1-x1))))+x1*log((exp(Beta0+Beta1*z1)/(1
  +exp(Beta0+Beta1*z1))))+(n1-x1)*log(1-(exp(Beta0+Beta1*z1)/(1+exp(Beta0+Beta1*z1))))
  +log((factorial(n2)/(factorial(x2)*factorial(n2-x2))))+x2*log((exp(Beta0+Beta1*z2)/(1
  +exp(Beta0+Beta1*z2))))+(n2-x2)*log(1-(exp(Beta0+Beta1*z2)/(1+exp(Beta0+Beta1*z2))))
  +log((factorial(n3)/(factorial(x3)*factorial(n3-x3))))+x3*log((exp(Beta0+Beta1*z3)/(1
  +exp(Beta0+Beta1*z3))))+(n3-x3)*log(1-(exp(Beta0+Beta1*z3)/(1+exp(Beta0+Beta1*z3))))
  +log((factorial(n4)/(factorial(x4)*factorial(n4-x4))))+x4*log((exp(Beta0+Beta1*z4)/(1
  +exp(Beta0+Beta1*z4))))+(n4-x4)*log(1-(exp(Beta0+Beta1*z4)/(1+exp(Beta0+Beta1*z4))))
}

# Plot of the function
pB0 <- seq(-6,-4,length=101)
pB1 <- seq(0.05,0.09,length=101)
logll.B0.B1 <- outer(pB0,pB1,function(B0,B1){print.logll(B0,B1)})
windows()
contour(pB0, pB1, logll.B0.B1, nlevels=100)
```

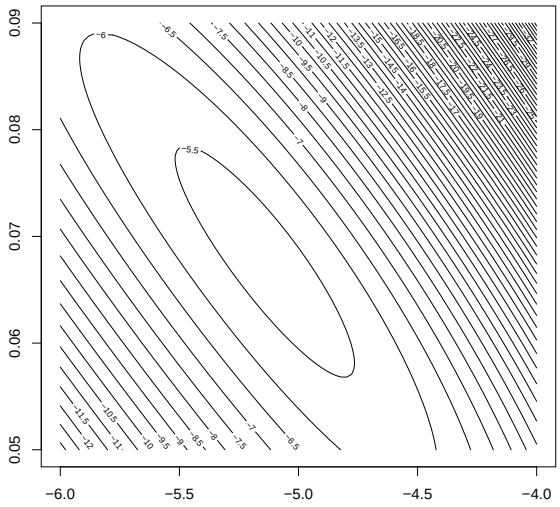


Figure 1: Likelihood function

A maximum is observed at a point around $-5$ and 0.07. In order to use the function **optim** the decision variables must be defined as a numeric vector. Hence, a slight modification must be introduced:

```
logll <-function(Beta0Beta1,x1=0,x2=2,x3=7,x4=3,z1=7,z2=14,z3=27,z4=57,n1=55,n2=157...
    Beta0 = Beta0Beta1[1]
    Beta1 = Beta0Beta1[2]

    -1*(log((factorial(n1)/(factorial(x1)*factorial(n1-x1))))+x1*log((exp(Beta0+...
}
```

Notice that the return is multiplied by minus one. This way we face a minimization problem. The same result can be achieved passing an argument to the function argument.

## 2.2 Function Scores

In order to derive the gradient of the log-likelihood equation we know that

$$\frac{\partial \mathcal{L}}{\partial \beta_i} = \sum_{i=1}^{4} \left( \frac{\frac{\partial}{\partial \beta_i} p_i}{p_i} - \frac{\frac{\partial}{\partial \beta_i}(1-p_i)}{1-p_i} \right) \cdot x_i + \frac{\frac{\partial}{\partial \beta_i}(1-p_i)}{1-p_i} \cdot n_i \tag{1}$$

Hence, we can define the following $2 \times 4$ matrix

$$Mij = \frac{1}{p_j} \cdot \frac{\partial}{\partial \beta_{i-1}} p_j + \frac{1}{1-p_j} \cdot \frac{\partial}{\partial \beta_{i-1}} p_j \tag{2}$$

Let us also define the $2 \times 1$ matrix $N$

$$N_i = \sum_{k=1}^{4} -\frac{1}{1-p_k} \cdot \frac{\partial}{\partial \beta_{i-1}} p_k \cdot n_k \tag{3}$$

And the gradient can be stated as

$$\bigtriangledown \mathcal{L} = M \cdot \underline{x} + N, \quad t(\underline{x}) = [x_1 \ x_2 \ x_3 \ x_4] \tag{4}$$

```
1
2   grad.ll <- function(Beta0Beta1,x1=0,x2=2,x3=7,x4=3,z1=7,z2=14,z3=27,z4=57,n1=55,n2...
3         Beta0 = Beta0Beta1[1]
4         Beta1 = Beta0Beta1[2]
5
6           p1 = exp(Beta0+Beta1*z1)/(1+exp(Beta0+Beta1*z1))
7           p2 = exp(Beta0+Beta1*z2)/(1+exp(Beta0+Beta1*z2))
8           p3 = exp(Beta0+Beta1*z3)/(1+exp(Beta0+Beta1*z3))
9           p4 = exp(Beta0+Beta1*z4)/(1+exp(Beta0+Beta1*z4))
10
11          diff.p1.B0 = exp(Beta1*z1+Beta0)/(1+exp(Beta0+Beta1*z1))-exp(2*Beta1*z1+2*...
12          diff.p2.B0 = exp(Beta1*z2+Beta0)/(1+exp(Beta0+Beta1*z2))-exp(2*Beta1*z2+2*...
13          diff.p3.B0 = exp(Beta1*z3+Beta0)/(1+exp(Beta0+Beta1*z3))-exp(2*Beta1*z3+2*...
14          diff.p4.B0 = exp(Beta1*z4+Beta0)/(1+exp(Beta0+Beta1*z4))-exp(2*Beta1*z4+2*...
15
16          diff.p1.B1 = (z1*exp(Beta1*z1+Beta0))/(exp(Beta1*z1+Beta0)+1)-(z1*exp(2*Be...
17          diff.p2.B1 = (z2*exp(Beta1*z2+Beta0))/(exp(Beta1*z2+Beta0)+1)-(z2*exp(2*Be...
18          diff.p3.B1 = (z3*exp(Beta1*z3+Beta0))/(exp(Beta1*z3+Beta0)+1)-(z3*exp(2*Be...
19          diff.p4.B1 = (z4*exp(Beta1*z4+Beta0))/(exp(Beta1*z4+Beta0)+1)-(z4*exp(2*Be...
20
21          M11 = 1/p1*diff.p1.B0 + 1/(1-p1)*diff.p1.B0
22          M12 = 1/p2*diff.p2.B0 + 1/(1-p2)*diff.p2.B0
23          M13 = 1/p3*diff.p3.B0 + 1/(1-p3)*diff.p3.B0
24          M14 = 1/p4*diff.p4.B0 + 1/(1-p4)*diff.p4.B0
25
26          M21 = 1/p1*diff.p1.B1 + 1/(1-p1)*diff.p1.B1
27          M22 = 1/p2*diff.p2.B1 + 1/(1-p2)*diff.p2.B1
28          M23 = 1/p3*diff.p3.B1 + 1/(1-p3)*diff.p3.B1
29          M24 = 1/p4*diff.p4.B1 + 1/(1-p4)*diff.p4.B1
30
31          N1 = - 1/(1-p1)*diff.p1.B0*n1 - 1/(1-p2)*diff.p2.B0*n2 - 1/(1-p3)*diff.p3.B0...
32          N2 = - 1/(1-p1)*diff.p1.B1*n1 - 1/(1-p2)*diff.p2.B1*n2 - 1/(1-p3)*diff.p3.B1...
33
34          M = matrix(c(M11,M12,M13,M14,M21,M22,M23,M24),byrow=T,nrow=2)
35          N = matrix(c(N1,N2),nrow=2)
36
37          X = matrix(c(x1,x2,x3,x4),nrow=4)
38
39          return(-1*(M%*%X+N))
40        }
```

## 2.3 Optimization of ML function

In this section different optimization algorithms are considered for minimizing the minus logarithm of the likelihood function. The default solver is the Nelder-Mead algorithm, which does not consider gradients but only function evaluations. For the other algorithms, the gradient is entered as a parameter. If information of the optimization procedure is desired, the input parameter trace is to be used. With the Hessian parameter set to *TRUE* a numeric approximation at the optimum point is given. When it comes to the Conjugate Gradient, there is a specific command for choosing a step rule: type. The maximum number of iterations can be enlarged with the parameter *maxit*. All initial points are set to $(0, 0)$.

Notice that the Hessian needs not to be computed for these methods, and a cheaper approximation is used instead. In the previous exercises, where de Fisher Scoring algorithm was studied, the Hessian matrix was derived analytically for some cases.

The general purpose optimization algorithms assume the Hessian to be dramatically more expensive to compute than the gradient or the objective function. However, in convex problems with an easy-to-compute Hessian matrix other algorithms may be more efficient.

```
1
2  X <- data.frame(matrix(rep(0,20),nrow=5))
3          names(X) <- c('Nelder_Mead','BFGS','CG','L_BFGS_B')
4          rownames(X) <- c('counts fobj.','counts grad','sol.B0','sol.B1','obj.val')
5
6
7  # a.1) Nelder-Mead
8      opt1 <- optim(par = c(0,0),fn = logll, control = list(trace=1))
9      # a.2) BFGS
10     opt2 <- optim(par = c(0,0),fn = logll, gr = grad.ll, method = "BFGS",hessian=T)
11     # a.3) CG
12     opt3 <- optim(par = c(0,0),fn = logll, gr = grad.ll, method = "CG",
13                                          control = list(maxit=10000,type=3),hessian=T)
14     # a.4) L-BFGS-B
15     opt4 <- optim(par = c(0,0),fn = logll,gr = grad.ll,method = "L-BFGS-B",hessian=T)
16
17
18 # Solution
19
20 > X
21                          Nelder_Mead          BFGS            CG     L_BFGS_B
22 counts fobj.          81.00000000  46.00000000  9.335200e+04  22.00000000
23 counts grad                   NA  14.00000000  1.000100e+04  22.00000000
24 sol. B0              -5.13154108  -5.13246849 -5.132465e+00  -5.13246849
25 sol. B1               0.06767118   0.06769814  6.769804e-02   0.06769814
26 obj. val             -5.33021360  -5.33021250 -5.330212e+00  -5.33021250
```

All the algorithms attain a similar solution. The Conjugate Gradient (CG) is not able to satisfy its internal optimality tolerance and reaches the maximum number of iterations. The Nelder-Mead method needs to evaluate the objective function 81 times. the BFGS method reduces this number to 46 at the cost of evaluating the gradient 14 times. When the gradient is known, derivative-free methods are not recommended, especially when functions are non-convex. The BFGS variant evaluates 22 times both the gradient and the objective function. The selection of the method depends on the computational cost of evaluating the objective function and the gradient.

## 2.4 Covariance matrix estimation

As stated before, a numerical approximation to the Hessian can be obtained with **optim**. The inverse of this matrix is an approximation to the covariance matrix.

```
solve(opt2$hessian)
# Solution

> solve(opt2$hessian)
              [,1]          [,2]
[1,]  0.41785875 -0.0106192304
[2,] -0.01061923  0.0003449474
```

This result can be compared to the result obtained using the Fishers Information computed analytically. It is easy to compute the Fisher's Information matrix as the variance of the scores function.

$$Var(M \cdot \underline{x} + N) = M \cdot Var(\underline{x}) \cdot t(M) + 0 \tag{5}$$

```
Fisher.inf <- function(Beta0,Beta1,x1=0,x2=2,x3=7,x4=3,z1=7,z2=14,z3=27,z4=57...

    (...)
  v.X1 <- n1*p1*(1-p1)
  v.X2 <- n2*p2*(1-p2)
  v.X3 <- n3*p3*(1-p3)
  v.X4 <- n4*p4*(1-p4)

  Var.X <- diag(c(v.X1,v.X2,v.X3,v.X4))

  return(M%*%Var.X%*%t(M))

# Solution

> solve(Fisher.inf(-5.13154108,0.06767118))
              [,1]          [,2]
[1,]  0.41771424 -0.0106170089
[2,] -0.01061701  0.0003449409
```

As it was expected the Hessian provided with the function **optim** is a very good approximation to the analitically derived Fishers information matrix. Hence, it can be used to estimate the variance of the maximum likelihood estimators.

## 2.5 Bootstrap Covariance matrix estimation

In this section a parametric bootstrap procedure is used to estimate the variance of the estimators with a derivative-free method. Our Data is assumed to follow a binomial distribution with the parameters obtained maximizing the likelihood function. In each iteration the maximum likelihood estimator must be obtained with the derivative-free algorithm as we do not have an analytic expression.

```
      # c) estimacio mitjan ant bootstrap parametric
        # assumim les probabilitats son les que hem trobat (nomes tenim la mostra)
          B0 = X$L_BFGS_B[3]
          B1 = X$L_BFGS_B[4]

          set.seed(7234)
          s.B = 1000

          p1 <- exp(B0+B1*7)/(1+exp(B0+B1*7))
          p2 <- exp(B0+B1*14)/(1+exp(B0+B1*14))
          p3 <- exp(B0+B1*27)/(1+exp(B0+B1*27))
          p4 <- exp(B0+B1*57)/(1+exp(B0+B1*57))
```

```
13
14            res.b <- replicate(s.B,
15                  {
16                    # estimacio per metode bootstrap parametric
17                    x1.b <- sum(rbinom(55,1,p1))
18                    x2.b <- sum(rbinom(157,1,p2))
19                    x3.b <- sum(rbinom(159,1,p3))
20                    x4.b <- sum(rbinom(16,1,p4))
21
22                    opt.b <- optim(par = c(0,0),fn = logll,x1=x1.b,x2=x2.b,x3=x3.b,x4=x4.b)
23                    opt.b$par
24                  }
25                  )
26
27            mean(res.b[1,])
28            mean(res.b[2,])
29
30            estimate.var.b <-matrix(rep(0,4),nrow=2)
31            estimate.var.b[1,1] <- var(res.b[1,])
32            estimate.var.b[2,2] <- var(res.b[2,])
33            estimate.var.b[1,2] <- cov(res.b[1,],res.b[2,])
34            estimate.var.b[2,1] <- cov(res.b[1,],res.b[2,])
35
36  # Solution
37  > estimate.var.b
38                                [,1]           [,2]
39                      [1,]  0.51914491 -0.0136233721
40                      [2,] -0.01362337  0.0004560286
```

The covariance matrix obtained is similar to the result obtained in the previous section. Bootstrap is a possibility of estimating the variance of the maximum likelihood estimators if the gradient is unknown or very difficult to compute. It has to be taken into account that the number of function evaluations of the derivative-free method is multiplied by the number of bootstrap samples.

# 3 Poisson GLM assumption

## 3.1 Algorithm deduction

The density function of a Poisson distribution is

$$f_{x_i} = \frac{\lambda_i^{x_i}}{x_i!} \cdot e^{-\lambda_i} \tag{6}$$

Then, the likelyhood function can be derived as

$$l = \prod_{i=1}^{n} \frac{\lambda_i^{x_i}}{x_i!} \cdot e^{-\lambda_i} \tag{7}$$

And the log-likelihood function

$$\mathcal{L} = \sum_{i=1}^{n} log\left(\frac{\lambda_i^{x_i}}{x_i!} \cdot e^{-\lambda_i}\right) = \sum_{i=1}^{n} x_i \cdot log(\lambda_i) - log(x_i!) - \lambda_i \tag{8}$$

If we substitute

$$\mathcal{L} = \sum_{i=1}^{n} x_i \cdot (B_0 + B_1 \cdot n_i + B_2 \cdot z_i) - log(x_i!) - e^{B_0 + B_1 \cdot n_i + B_2 \cdot z_i} \tag{9}$$

We then derive in order to get the gradient

$$\frac{\partial \mathcal{L}}{\partial B_0} = \sum_{i=1}^{n} x_i - \sum_{i=1}^{n} e^{B_0 + B_1 \cdot n_i + B_2 \cdot z_i} = \sum_{i=1}^{n} x_i - \sum_{i=1}^{n} \lambda_i \tag{10}$$

$$\frac{\partial \mathcal{L}}{\partial B_1} = \sum_{i=1}^{n} x_i \cdot n_i - \sum_{i=1}^{n} e^{B_0 + B_1 \cdot n_i + B_2 \cdot z_i} \cdot n_i = \sum_{i=1}^{n} x_i \cdot n_i - \sum_{i=1}^{n} \lambda_i \cdot n_i \tag{11}$$

$$\frac{\partial \mathcal{L}}{\partial B_2} = \sum_{i=1}^{n} x_i \cdot z_i - \sum_{i=1}^{n} e^{B_0 + B_1 \cdot n_i + B_2 \cdot z_i} \cdot z_i = \sum_{i=1}^{n} x_i \cdot z_i - \sum_{i=1}^{n} \lambda_i \cdot z_i \tag{12}$$

This gradient can be expressed in a matrix form as

$$\bigtriangledown \mathcal{L} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ n_1 & n_2 & n_3 & n_4 \\ z_1 & z_2 & z_3 & z_4 \end{pmatrix} \cdot \begin{pmatrix} x_1 - \lambda_1 \\ x_2 - \lambda_2 \\ x_3 - \lambda_3 \\ x_4 - \lambda_4 \end{pmatrix} = X \cdot [\underline{x} - \underline{\lambda}] \tag{13}$$

And the Fisher's Information matrix

$$I_x = var(\bigtriangledown \mathcal{L}) = X \cdot \begin{pmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 \\ 0 & 0 & \lambda_3 & 0 \\ 0 & 0 & 0 & \lambda_4 \end{pmatrix} \cdot X^t = X \cdot W \cdot X^t \tag{14}$$

Now we formulate an iteration of the Fisher Scoring method

$$\begin{pmatrix} B_0^{k+1} \\ B_1^{k+1} \\ B_2^{k+1} \end{pmatrix} = \begin{pmatrix} B_0^{k} \\ B_1^{k} \\ B_2^{k} \end{pmatrix} + (X \cdot W \cdot X^t)^{-1} \cdot X \cdot W \cdot (W^{-1} \cdot (x - \lambda)) \tag{15}$$

And after some algebra and grouping terms we obtain

$$(X \cdot W \cdot X^t)^{-1} \cdot (X \cdot W) \cdot ((X^t \cdot \underline{B}) + (W^{-1} \cdot (\underline{x} - \underline{\lambda}))) \tag{16}$$

When we have a model of the form

$$Y = X \cdot \beta + \varepsilon \tag{17}$$

where the variance of the error is not constant it is called Weighted Least Squares. The maximum likelyhood estimator of $\beta$ in those models is

$$\beta^{ML} = (X^t W X^t)^{-1} \cdot X^t \cdot W \cdot Y \tag{18}$$

Hence, if we make a variable change we can solve an iteration of the Fisher Scoring algorithm solving a WLS

$$Z = ((X^t \cdot \underline{B}) + (W^{-1} \cdot (\underline{x} - \underline{\lambda}))) \tag{19}$$

$$z_i = B_0 + n_i \cdot B_1 + z_i \cdot B_2 + \frac{x_i - \lambda_i}{\lambda_i} \tag{20}$$

### 3.2 Algorithm implementation

In this section the algorithm is implemented in R following the definitions of the previous section.

```
# Algorisme
        B_0 <- NULL
        B_1 <- NULL
        B_2 <- NULL

        B_0[1] = -1
        B_1[1] = 0.01
        B_2[1] = 0.01

        n.iter = 10

        lambda <- matrix(0,nrow=n.iter,ncol=4)
        z <- matrix(0,nrow=n.iter,ncol=4)
        v <- matrix(0,nrow=n.iter,ncol=4)

        for (k in 1:n.iter){

          lambda[k,] = exp(B_0[k]+n*B_1[k]+z_t*B_2[k])
          z[k,] = B_0[k] + B_1[k]*n + B_2[k]*z_t + (x - lambda[k,])/lambda[k,]
          v[k,] = lambda[k,]

          z_p <- z[k,]
          v_p <- v[k,]
          model <- lm(z_p~n+z_t,weights = v_p)

          B_0[k+1] = model$coefficients[1]
          B_1[k+1] = model$coefficients[2]
          B_2[k+1] = model$coefficients[3]
        }
```

### 3.3 Algorithm execution

```
# Solution of the algorithm
> lambda
                [,1]      [,2]       [,3]       [,4]
      [1,] 0.68386141 2.033991  2.363161  0.7633795
      [2,] 0.17451595 2.934334 12.500842 17.8838567
      [3,] 0.09544849 1.982917  8.176460  7.7601505
      [4,] 0.06697290 1.815442  7.231743  4.1793774
      [5,] 0.05728491 1.823928  7.158590  3.1227954
      [6,] 0.05563055 1.826722  7.155692  2.9660531
      [7,] 0.05558569 1.826784  7.155634  2.9619994
      [8,] 0.05558566 1.826784  7.155634  2.9619967
      [9,] 0.05558566 1.826784  7.155634  2.9619967
     [10,] 0.05558566 1.826784  7.155634  2.9619967

> cbind(B_0,B_1,B_2)
             B_0         B_1        B_2
      [1,] -1.000000 0.01000000 0.0100000
      [2,] -3.617081 0.02023144 0.1083733
      [3,] -4.325343 0.02250134 0.1055144
      [4,] -4.813135 0.02532187 0.1024236
      [5,] -5.051644 0.02699542 0.1010254
      [6,] -5.096938 0.02731128 0.1008279
      [7,] -5.098184 0.02731983 0.1008234
```

```
      [8,] -5.098185 0.02731984 0.1008233
      [9,] -5.098185 0.02731984 0.1008233
     [10,] -5.098185 0.02731984 0.1008233

# resultats de la funcio glm
glm(x - n + z_t, family = poisson)

# Solution

> glm(x - n + z_t, family=poisson)

Coefficients:
(Intercept)            n          z_t
   -5.09818      0.02732      0.10082
```

After 8 iterations an optimum is attained. The result is the same with our implementation and with the function glm of R.