To appear in the *Journal of Applied Statistics*
Vol. 00, No. 00, Month 20XX, 1–11

SUPPLEMENTAL MATERIAL

The Poisson-exponential Model for Recurrent Event Data: An Application to Bowel Motility Data

Francisco Louzada[a], Márcia A. C. Macera[b*] and Vicente G. Cancho[a]

[a] *Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, Caixa Postal 668, São Carlos, SP, 13560-970, Brasil*;
[b] *Departamento de Estatística, Universidade Federal de São Carlos, Via Washington Luís, km 235, Caixa Postal 676, São Carlos, SP, 13565-905, Brasil*

(*Received 00 Month 20XX; accepted 00 Month 20XX*)

In this supplemental material we provide further details on the observed information matrix and codes of the simulation and case study described in the main body of the paper. Appendix A displays the components of the Fisher information matrix of Section 3. Appendix B provides additional functions required for the simulation and case study codes. The code to reproduce Figure 1 of Section 2 is provided in Appendix C. The codes to reproduce the results of the simulation study of Section 4 and the results of the case study of Section 5 are presented in Appendices D and E, respectively.

Appendix A. Calculation details for the observed information matrix of Section 3

The components of the observed information matrix, $I(\boldsymbol{\vartheta})$, are derived in the form

$$I_{11} = -\frac{\partial^2 \ell(\boldsymbol{\vartheta})}{\partial \alpha^2} = \sum_{i=1}^{n}\sum_{j=1}^{m_i} c_{ij}\left\{\alpha^{-2} + \theta(t_{i,j-1}+w)^2 \mathrm{e}^{-\alpha(t_{i,j-1}+w)}\right\}$$

$$- \frac{(1-c_{ij})\theta(t_{i,j-1}+w)^2 \mathrm{e}^{-\alpha(t_{i,j-1}+w)}}{\mathrm{e}^{\theta \mathrm{e}^{-\alpha(t_{i,j-1}+w)}}-1}\left[1 - \frac{\theta \mathrm{e}^{-\alpha(t_{i,j-1}+w)}}{1-\mathrm{e}^{-\theta \mathrm{e}^{-\alpha(t_{i,j-1}+w)}}}\right]$$

$$+ \frac{\theta(t_{i,j-1})^2 \mathrm{e}^{-\alpha t_{i,j-1}}}{\mathrm{e}^{\theta \mathrm{e}^{-\alpha t_{i,j-1}}}-1}\left[1 - \frac{\theta \mathrm{e}^{-\alpha t_{i,j-1}}}{1-\mathrm{e}^{-\theta \mathrm{e}^{-\alpha t_{i,j-1}}}}\right].$$

*Corresponding author. Email: marciamacera@gmail.com

$$I_{12} = -\frac{\partial^2 \ell(\boldsymbol{\vartheta})}{\partial\theta\partial\alpha} = I_{21} = \sum_{i=1}^{n}\sum_{j=1}^{m_i} -c_{ij}(t_{i,j-1}+w)\mathrm{e}^{-\alpha(t_{i,j-1}+w)}$$

$$+ \frac{(1-c_{ij})(t_{i,j-1}+w)\mathrm{e}^{-\alpha(t_{i,j-1}+w)}}{\mathrm{e}^{\theta\mathrm{e}^{-\alpha(t_{i,j-1}+w)}}-1}\left[1 - \frac{\theta\mathrm{e}^{-\alpha(t_{i,j-1}+w)}}{1-\mathrm{e}^{-\theta\mathrm{e}^{-\alpha(t_{i,j-1}+w)}}}\right]$$

$$- \frac{(t_{i,j-1})\mathrm{e}^{-\alpha t_{i,j-1}}}{\mathrm{e}^{\theta\mathrm{e}^{-\alpha t_{i,j-1}}}-1}\left[1 - \frac{\theta\mathrm{e}^{-\alpha t_{i,j-1}}}{1-\mathrm{e}^{-\theta\mathrm{e}^{-\alpha t_{i,j-1}}}}\right].$$

$$I_{22} = -\frac{\partial^2 \ell(\boldsymbol{\vartheta})}{\partial\theta^2} = \sum_{i=1}^{n}\sum_{j=1}^{m_i} c_{ij}\theta^{-2} + (1-c_{ij})\left[\frac{\mathrm{e}^{\theta\mathrm{e}^{-\alpha(t_{i,j-1}+w)}}(\mathrm{e}^{-\alpha(t_{i,j-1}+w)})^2}{(\mathrm{e}^{\theta\mathrm{e}^{-\alpha(t_{i,j-1}+w)}}-1)^2}\right]$$

$$- \frac{\mathrm{e}^{\theta\mathrm{e}^{-\alpha t_{i,j-1}}}(\mathrm{e}^{-\alpha t_{i,j-1}})^2}{(\mathrm{e}^{\theta\mathrm{e}^{-\alpha t_{i,j-1}}}-1)^2}.$$

Appendix B. Additional functions required for the codes in Appendices C, D and E

```
# function to generate artificial data
GenData <- function(alpha, theta, n, m, p) {
        # alpha, theta: parameter values; n: sample size; m: number of recurrences
            ; p: percent of censoring
        temp <- matrix(0, ncol=m+1, nrow=n); w <- matrix(ncol=m, nrow=n); cens <-
            matrix(1, nrow=n, ncol=m)
        for (i in 1:n) {
                sum_aux = 0; j = 1
                while(TRUE) {
                        u <- runif(1,0,1)
                        if (theta == 0) {
                                w[i,j] <- (-(1/alpha)*log(u))
                        }
                        else {
                                w[i,j] <- ((log(theta)-alpha*(temp[i,j])-log(-log(
                                    u+((1-u)*exp(-theta*exp(-alpha*temp[i,j])))))
                                    /alpha)
                        }
                        if ((w[i,j] == Inf)||(w[i,j] == 0)||(is.na(w[i,j]))) j=1
                        temp[i,j+1] <- temp[i,j] + w[i,j]
                        sum_aux = temp[i,j+1]
                        if ((j == m)) break
                        j = j+1
                }
        }
        if (m != 2) {
                ind <- sample(1:n, round(p*m))
                cens[ind,m] <- 0
        }
        return(list(temp=temp, w=w, cens=cens))
}
# log-likelihood function
param = numeric(0)
fr <- function(param, cens, temp=0, w=0, n, h0=FALSE) {
        # param: vector of parameter values; cens: censoring vector; temp:
            calendar times vector; w: gap times vector; n: sample size
        # h0 = TRUE (considering the hypothesis H0: theta=0)
```

```
        vetsoma = 0
        if (h0 == FALSE) {
                p1 <- exp(param[1])  # alpha
                p2 <- exp(param[2])  # theta
                vetsoma = lapply(1:n, function(k) {tam=length(temp[[k]]); aux <-
                    (-cens[[k]]*(log(p1)+log(p2)-p1*(temp[[k]][2:tam])-p2*exp(-p1*
                    (temp[[k]][2:tam]))))
                  -(1-cens[[k]])*log(1-exp(-p2*exp(-p1*(temp[[k]][2:tam]))))+log
                      (1-exp(-p2*exp(-p1*(temp[[k]][1:(tam-1)])))))); sum(aux)})
        }
        else {
                p <- exp(param)  # alpha
                vetsoma = lapply(1:n, function(k) {aux <- (-cens[[k]]*log(p)+p*w[[
                    k]]); sum(aux)})
        }
        llike <- sum(unlist(vetsoma))
        return(llike)
}
# function to calculate the observed information matrix
hess <- function(param, cens, temp, n) {
        # param: vector of parameter values; cens: censoring vector; temp:
            calendar times vector; n: sample size
        aux11 <-0; aux12 <-0; aux22 <-0
        p1 <- param[1]  # alpha
        p2 <- param[2]  # theta
        # second derivative with respect to alpha
        aux11 = lapply(1:n, function(k) {tam=length(temp[k,]); hessiA = (cens[[k]]
            *((1/(p1^2))+p2*((temp[[k]][2:tam])^2)*exp(-p1*temp[[k]][2:tam]))
                -(((1-cens[[k]])*p2*((temp[[k]][2:tam])^2)*exp(-p1*temp[[k]][2:
                    tam])/(exp(p2*exp(-p1*temp[[k]][2:tam]))-1))*(1-
                        (p2*exp(-p1*temp[[k]][2:tam])/(1-exp(-p2*exp(-p1*temp[[k
                            ]][2:tam])))))+((p2*((temp[[k]][1:(tam-1)])^2)*exp(-
                            p1*temp[[k]][1:(tam-1)])
                        /(exp(p2*exp(-p1*temp[[k]][1:(tam-1)]))-1))*(1-(p2*exp(-
                            p1*temp[[k]][1:(tam-1)])/(1-exp(-p2*exp(-p1*temp[[k
                            ]][1:(tam-1)])))))))); sum(hessiA)})
        a11 <- sum(unlist(aux11))
        # second derivative of theta with respect to alpha
        aux12 = lapply(1:n, function(k) {tam=length(temp[k,]); hessiTA = (-cens[[k
            ]]*temp[[k]][2:tam]*exp(-p1*temp[[k]][2:tam])
                +(((1-cens[[k]])*temp[[k]][2:tam]*exp(-p1*temp[[k]][2:tam])/(exp(
                    p2*exp(-p1*temp[[k]][2:tam]))-1))*(1-
                        (p2*exp(-p1*temp[[k]][2:tam])/(1-exp(-p2*exp(-p1*temp[[k
                            ]][2:tam])))))-((temp[[k]][1:(tam-1)]*
                        exp(-p1*temp[[k]][1:(tam-1)])/(exp(p2*exp(-p1*temp[[k
                            ]][1:(tam-1)]))-1))*(1-(p2*exp(-p1*temp[[k]][1:(tam
                            -1)])/
                        (1-exp(-p2*exp(-p1*temp[[k]][1:(tam-1)])))))))); sum(
                            hessiTA)})
        a12 <- sum(unlist(aux12))
        # second derivative with respect to theta
        aux22 = lapply(1:n, function(k) {tam=length(temp[k,]); hessiT = ((cens[[k
            ]]*(p2^(-2)))+((1-cens[[k]])*
                (exp(p2*exp(-p1*temp[[k]][2:tam]))*exp(-2*p1*temp[[k]][2:tam])/(
                    exp(p2*exp(-p1*temp[[k]][2:tam]))-1)^2))
                        -(exp(p2*exp(-p1*temp[[k]][1:(tam-1)]))*exp(-2*p1*temp[[k
                            ]][1:(tam-1)])/(exp(p2*exp(-p1*temp[[k]][1:(tam-1)]))
                            -1)^2)); sum(hessiT)})
        a22 <- sum(unlist(aux22))
        matrix(c(a11, a12, a12, a22), nrow=2, byrow=T)
}
# function to calculate the observed information matrix (HPP model)
hessHPP <- function(param, cens, n) {
        # param: vector of parameter value; cens: censoring vector; n: sample size
        sm <-0
        p <- param  # alpha
        sm = lapply(1:n, function(k) {hessA <- (cens[[k]]*(p^(-2))); sum(hessA)})
```

```
        aa <- sum(unlist(sm))
        return(aa)
}
# function to calculate the Score statistic
Test.score <- function(param, temp, cens, n) {
        # param: vector of parameter values; temp: calendar times vector; cens:
            censoring vector; n: sample size
        aux1=0; aux2=0; A=0; B=0
        p1 <- param[1]  # alpha
        p2 <- param[2]  # theta
        # score function of theta
        aux1 = lapply(1:n, function(k) {tam=length(temp[k,]); score = (cens[[k]]*
            ((p2^(-1))-exp(-p1*temp[[k]][2:tam]))
        +(((1-cens[[k]])*exp(-p1*temp[[k]][2:tam]))/(exp(p2*exp(-p1*temp[[k]][2:
            tam]))-1))
                -((exp(-p1*temp[[k]][1:(tam-1)]))/(exp(p2*exp(-p1*temp[[k]][1:(tam
                    -1)]))-1))); sum(score)})
        a1 = sum(unlist(aux1))
        # second derivative with respect to theta
        aux2 = lapply(1:n, function(k) {tam=length(temp[k,]); hessiT = ((cens[[k]]
            *(p2^(-2)))+((1-cens[[k]])*
                (exp(p2*exp(-p1*temp[[k]][2:tam]))*exp(-2*p1*temp[[k]][2:tam])/(
                    exp(p2*exp(-p1*temp[[k]][2:tam]))-1)^2))
                -(exp(p2*exp(-p1*temp[[k]][1:(tam-1)]))*exp(-2*p1*temp[[k]][1:(tam
                    -1)])/(exp(p2*exp(-p1*temp[[k]][1:(tam-1)]))-1)^2)); sum(
                    hessiT)})
        a2 <- sum(unlist(aux2))
        # second derivative of theta with respect to alpha
        A = lapply(1:n, function(k) {tam=length(temp[k,]); hessiTA = ((-cens[[k]]*
            temp[[k]][2:tam]*exp(-p1*temp[[k]][2:tam]))
            +(((1-cens[[k]])*temp[[k]][2:tam]*exp(-p1*temp[[k]][2:tam])/(exp(p2*
                exp(-p1*temp[[k]][2:tam]))-1))*(1-
                (p2*exp(-p1*temp[[k]][2:tam])/(1-exp(-p2*exp(-p1*temp[[k]][2:tam])
                    )))))-((temp[[k]][1:(tam-1)]
                *exp(-p1*temp[[k]][1:(tam-1)])/(exp(p2*exp(-p1*temp[[k]][1:(tam-1)
                    ]))-1))*(1-(p2*exp(-p1*temp[[k]][1:(tam-1)])
                /(1-exp(-p2*exp(-p1*temp[[k]][1:(tam-1)])))))))); sum(hessiTA)})
        a3 <- sum(unlist(A))
        # second derivative with respect to alpha
        B = lapply(1:n, function(k) {tam=length(temp[k,]); hessiA = (cens[[k]]*((1
            /(p1^2))+p2*((temp[[k]][2:tam])^2)*exp(-p1*temp[[k]][2:tam]))
            -(((1-cens[[k]])*p2*((temp[[k]][2:tam])^2)*exp(-p1*temp[[k]][2:tam])/(
                exp(p2*exp(-p1*temp[[k]][2:tam]))-1))*(1-
                (p2*exp(-p1*temp[[k]][2:tam])/(1-exp(-p2*exp(-p1*temp[[k]][2:tam])
                    )))))+((p2*((temp[[k]][1:(tam-1)])^2)
                *exp(-p1*temp[[k]][1:(tam-1)])/(exp(p2*exp(-p1*temp[[k]][1:(tam-1)
                    ]))-1))*(1-(p2*exp(-p1*temp[[k]][1:(tam-1)])
                /(1-exp(-p2*exp(-p1*temp[[k]][1:(tam-1)])))))); sum(hessiA)})
        b1 <- sum(unlist(B))
        Z.test <- (a1^2)/(a2-((a3^2)*(1/b1)))
        return(Z.test)
}
```

Appendix C. Code to reproduce Figure 1 of Section 2

```
# generate times to the rate function plot
time_rf = GenData(5.8, 35, 1, 50, 0.3)$temp
x = time_rf[2:51]
# rate function
rate = function(alpha, theta, x) {
        # alpha, theta: parameter values; x: recurrence times
        h = (theta*alpha*exp(-alpha*x))/(exp(theta*exp(-alpha*x))-1)
        return(h)
}
# different values of lambda
theta = c(0.2, 1, 3, 6)
```

```
alpha = 1
# initializing an auxiliary variable
aux = matrix(0, nrow=length(theta), ncol=length(x))
# calculate the rate function for each value of lambda
for (i in 1:length(theta)) {
        aux[i,] = rate(alpha, theta[i], x)
}
ptmaxY = max(aux); ptminY = min(aux)
plot(x, aux[1,], ylim=c(ptminY,ptmaxY), type='l', ylab="Rate function", xlab="
    Recurrence times", lty=1, lwd=1.7, col=1)
for (i in 2:length(theta)){
        lines(x, aux[i,], ylim=c(ptminY,ptmaxY), type='l', col=i, lty=i+1, lwd
            =1.5)
}
legend(5.5,0.5, legend=c(expression(paste(plain(theta) == 0.2)), expression(paste(
    plain(theta) == '1.0')), expression(paste(plain(theta) == '3.0')),
        expression(paste(plain(theta) == '6.0'))), lty=c(1,3,4,5), lwd=c
            (1.7,1.5,1.5,1.5), col=c(1,2,3,4), cex=0.9)
```

Appendix D. Code to reproduce the results of the simulation study discussed in Section 4

```
# loading library
require(MASS)
# sample size, number of recurrences and percent of censoring
n <- 30  # n = 30,50,100
m <- 2  # m = 2,5,7,15
p <- 0.3  # p = 0.25(for m = 5) and p = 0.30(for m = 7,15)
# alpha and theta parameter values
alpha <- 3
theta <- 4  # theta = 0.75,2,4

# Note: simulation is performed for each of the 36 settings
# the simulations take time!

result <- data.frame(0,0,0,0,0,0,0,0,0)
names(result) <- c("Alpha","VarAlpha","LI","LS","Theta","VarTheta","LI","LS","
    Covars")
# number of simulations
simul <- 1000
## do simulations ##
set.seed(2143)
s=1
while (s <= simul) {
        # generate the data
        dados <- GenData(alpha=alpha, theta=theta, n=n, m=m, p=p)
        times = lapply(split(dados$temp, f=1:nrow(dados$temp)), matrix, ncol=ncol(
            dados$temp), nrow=1)
        censor = lapply(split(dados$cens, f=1:nrow(dados$cens)), matrix, ncol=ncol
            (dados$cens), nrow=1)
        # fit model
        # should be considered as initial values for the model adjustment the
            approximate values of "log(alpha)" and "log(theta)"
        otim <- optim(par=c(1.098,1.39), method="BFGS", fn=fr, hessian=TRUE, cens=
            censor, temp=times, n=n, control=list(reltol=1e-5))
        # compute the observed information matrix
        FisherInf <- hess(exp(otim$par), cens=censor, temp=times, n=n)
        if (is.nan(sum(FisherInf))) {
        }
        else {
                # compute the variance and covariance from the information matrix
                aux <- ginv(FisherInf)
                vetvars <- diag(aux);  covars <- aux[1,2]
                if (is.nan(sqrt(vetvars[1])) || is.nan(sqrt(vetvars[2])) ) {
                }
                else {
```

```
                        # compute the confidence intervals for parameters of the
                            model
                        IC <- matrix(c(exp(otim$par)-1.96*sqrt(vetvars), exp(otim$
                            par)+1.96*sqrt(vetvars)), ncol=2, byrow=F)
                        # get the results for parameter alpha
                        result[s,1] <- exp(otim$par[1])
                        result[s,2] <- vetvars[1]
                        result[s,3] <- IC[1,1]
                        result[s,4] <- IC[1,2]
                        # get the results for parameter theta
                        result[s,5] <- exp(otim$par[2])
                        result[s,6] <- vetvars[2]
                        result[s,7] <- IC[2,1]
                        result[s,8] <- IC[2,2]
                        # get covariance
                        result[s,9] <- covars
                        s <- s+1
                }
        }
}
# Results of Tables 1 and 2
# mean and standard erros of 1000 MLEs
medpar <- c(mean(result[,1]), mean(result[,5]))
sd_IO <- c(mean(sqrt(resultados[,2])), mean(sqrt(resultados[,6])))
# simulated values of variance and covariance
var_est <- c((sum((result[,1]-medpar[1])^2)/simul), (sum((result[,5]-medpar[2])^2)
    /simul))
covar_est <- cov(result[,1], result[,5])
# approximated values of variance and covariance from the observed information
var_IO <- c(mean(result[,2]), mean(result[,6]))
covar_IO <- mean(result[,9])
# coverage probabilities
L1 <- length(which(result[,3] > alpha))/simul
U1 <- length(which(result[,4] < alpha))/simul
PC_alph <- 1-(L1+U1)
L2 <- length(which(result[,7] > theta))/simul
U2 <- length(which(result[,8] < theta))/simul
PC_thet <- 1-(L2+U2)

## Likelihood ratio (LRS) test and Score test ##
# type I error
# sample size, number of recurrences and percent of censoring
n <- 30  # n = 30,50,100
m <- 2  # m = 2,5,7,15
p <- 0.3  # p = 0.25(for m = 5) and p = 0.30(for m = 7,15)
# number of simulations
simul <- 1000
# initializing auxiliary variables
aux0=numeric(0); vet0=numeric(0); al0=numeric(0)
# Results of Table 3
# LRS test
set.seed(2143)
for (s in 1:simul) {
        # generate data considering H0: theta=0
        data_lr = GenData(alpha=3, theta=0, n=n, m=m, p=p)
        times_lr = lapply(split(data_lr$temp, f=1:nrow(data_lr$temp)), matrix,
            ncol=ncol(data_lr$temp), nrow=1)
        censor_lr = lapply(split(data_lr$cens, f=1:nrow(data_lr$cens)), matrix,
            ncol=ncol(data_lr$cens), nrow=1)
        w_lr = lapply(split(data_lr$w, f=1:nrow(data_lr$w)), matrix, ncol=ncol(
            data_lr$w), nrow=1)
        # fit complete model
        otimH1 <- optim(par=c(1.098, 0.005), method="BFGS", fn=fr, cens=censor_lr,
             temp=times_lr, n=n, control=list(reltol=1e-5))
        llike1 <- -fr(otimH1$par, cens=data_lr$cens, temp=data_lr$temp, n=n)
        # fit submodel (considering H0: theta=0)
```

```
        otimH0 <- optim(par=1.098, method="BFGS", fn=fr, cens=censor_lr, w=w_lr, n
            =n, h0=T, control=list(reltol=1e-5))
        llike0 <- -fr(otimH0$par, cens=censor_lr, w=w_lr, n=n, h0=T)
        # compute the LR statistic
        LRS0 <- 2*(llike1-llike0)
        aux0[s] <- LRS0
}
# empirical proportions of type I error for LRS test
rejec = length(which(aux0 > 2.71))/simul
# Score test
set.seed(2143)
for (h in 1:simul) {
        # generate data considering H0: lambda=0
        data_sc = GenData(alpha=3, theta=0, n=n, m=m, p=p)
        times_sc = lapply(split(data_sc$temp, f=1:nrow(data_sc$temp)), matrix,
            ncol=ncol(data_sc$temp), nrow=1)
        censor_sc = lapply(split(data_sc$cens, f=1:nrow(data_sc$cens)), matrix,
            ncol=ncol(data_sc$cens), nrow=1)
        w_sc = lapply(split(data_sc$w, f=1:nrow(data_sc$w)), matrix, ncol=ncol(
            data_sc$w), nrow=1)
        # fit submodel (considering H0: theta=0)
        otimH0_sc <- optim(par=1.098, method="BFGS", fn=fr, cens=censor_sc, w=w_sc
            , n=n, h0=T, control=list(reltol=1e-5))
        al0[h] = exp(otimH0_sc$par)
        th0 = 0.02
        # compute the score statistic (considering H0: theta=0)
        TS0 <- Test.score(param=c(al0[h], th0), temp=times_sc, cens=censor_sc, n=n
            )
        vet0[h] <- TS0
}
# empirical proportions of type I error for score test
rejec1 <- length(which(vet0 > 3.84))/simul

# power of test
# sample size, number of recurrences and percent of censoring
n <- 30  # n = 30,50,100
m <- 2  # m = 2,5,7,15
p <- 0.3  # p = 0.25(for m = 5) and p = 0.30(for m = 7,15)
# theta parameter values
theta <- 0.75 # theta = 0.75,1.5,3
# number of simulations
simul <- 1000
# initializing auxiliary variables
aux1=numeric(0); vet1=numeric(0); al1=numeric(0)
# Results of Table 4
# LRS test
set.seed(2143)
for (j in 1:simul) {
        # generate data considering the complete model
        data_lr1 = GenData(alpha=3, theta=theta, n=n, m=m, p=p)
        times_lr1 = lapply(split(data_lr1$temp, f=1:nrow(data_lr1$temp)), matrix,
            ncol=ncol(data_lr1$temp), nrow=1)
        censor_lr1 = lapply(split(data_lr1$cens, f=1:nrow(data_lr1$cens)), matrix,
             ncol=ncol(data_lr1$cens), nrow=1)
        w_lr1 = lapply(split(data_lr1$w, f=1:nrow(data_lr1$w)), matrix, ncol=ncol(
            data_lr1$w), nrow=1)
        # fit complete model
        # should be considered as initial values for the model adjustment the
            approximate values of "log(alpha)" and "log(theta)"
        otimC <- optim(par=c(1.098, log(theta)), method="BFGS", fn=fr, cens=censor
            _lr1, temp=times_lr1, n=n, control=list(reltol=1e-5))
        llikeC <- -fr(otimC$par, cens=censor_lr1, temp=times_lr1, n=n)
        # fit submodel (considering H0: lambda=0)
        otimR <- optim(par=1.098, method="BFGS", fn=fr, cens=censor_lr1, w=w_lr1,
            n=n, h0=T, control=list(reltol=1e-5))
        llikeR <- -fr(otimR$par, cens=censor_lr1, w=w_lr1, n=n, h0=T)
        # compute the LRS statistic
```

```
        LRS1 <- 2*(llikeC-llikeR)
        aux1[j] <- LRS1
}
# empirical power of the LR test
pod = length(which(aux1 > 2.71))/simul
# Score test
set.seed(2143)
for (k in 1:simul) {
        # generate data considering the complete model
        data_sc1 = GenData(alpha=3, theta=theta, n=n, m=m, p=p)
        times_sc1 = lapply(split(data_sc1$temp, f=1:nrow(data_sc1$temp)), matrix,
            ncol=ncol(data_sc1$temp), nrow=1)
        censor_sc1 = lapply(split(data_sc1$cens, f=1:nrow(data_sc1$cens)), matrix,
             ncol=ncol(data_sc1$cens), nrow=1)
        w_sc1 = lapply(split(data_sc1$w, f=1:nrow(data_sc1$w)), matrix, ncol=ncol(
            data_sc1$w), nrow=1)
        # fit submodel (considering H0: lambda=0)
        otimH01 <- optim(par=1.098, method="BFGS", fn=fr, cens=censor_sc1, w=w_sc1
            , n=n, h0=T, control=list(reltol=1e-5))
        al1[k] <- exp(otimH01$par)
        th1 <- 0.07
        # compute the score statistic (considering H0: theta=0)
        TS1 <- Test.score(param=c(al1[k], th1), temp=times_sc1, cens=censor_sc1, n
            =n)
        vet1[k] <- TS1
}
# empirical power of the score test
pod1 = length(which(vet1 > 3.84))/simul

## Overall goodness of fit - Cox-Snell residuals ##
# loading libraries
require(gdata)
require(survival)
# sample size, number of recurrences and percent of censoring
n <- 30  # n = 30,50,100
m <- 2  # m = 2,5,7,15
p <- 0.3  # p = 0.25(for m = 5) and p = 0.30(for m = 7,15)
set.seed(2143)
# generate data
data_res <- GenData(alpha=3, theta=2, n=n, m=m, p=p)
temp = lapply(split(data_res$temp, f=1:nrow(data_res$temp)), matrix, ncol=ncol(
    data_res$temp), nrow=1)
cens = lapply(split(data_res$cens, f=1:nrow(data_res$cens)), matrix, ncol=ncol(
    data_res$cens), nrow=1)
# fit model
otim_res <- optim(par=c(1.098,0.69), method="BFGS", fn=fr, cens=cens, temp=temp, n
    =n, control=list(reltol=1e-5))
# get MLEs
a0=exp(otim_res$par[1]); t0=exp(otim_res$par[2])
# Cox-snell residuals calculated for each unit and each recurrence
cs = lapply(1:n, function(h) {tam=length(temp[[h]]); res = log((1-exp(-t0*exp(-a0*
    temp[[h]][1:(tam-1)])))/(1-exp(-t0*exp(-a0*temp[[h]][2:tam]))))})
cumcs = list(); cont = list()
for (m in 1:length(cs)) {
        cont[[m]] = 0
}
# get a cumulative residual at each recurrence (for each unit)
for(y in 1:length(cs)) {
        for(l in 1:(length(cs[[y]]))){
                cont[[y]][l+1] = cont[[y]][l]+cs[[y]][l]
    }
    cumcs[[y]] = cont[[y]][2:length(cont[[y]])]
}
# transform lists into a data.frame
cumcs2 = unlist(cumcs)
event = unmatrix(cens, byrow=T)
tam = unlist(lapply(cumcs,length))
```

```
id = rep(seq(1,length(tam)),tam)
# organize data to calculate the Nelson-Aalen estimate of the cumulative hazard
    function
conjunt = data.frame(id,event,cumcs2)
z=rep(0,length(conjunt[,1]))
conjunt=cbind(conjunt,z,z)
colnames(conjunt)=c("id","event","cumcs2","initial","final")
v = unique(conjunt$id)
tamv = length(v)
for (w in 1:tamv) {
        a = c(0,conjunt[conjunt$id==v[w],]$cumcs2)
        for (j in 1:(length(a)-1)) {
                conjunt[conjunt$id==v[w],][j,]$initial = a[j]
                conjunt[conjunt$id==v[w],][j,]$final = a[j+1]
        }
}
# Figure 2
# Nelson-Aalen estimate of the cumulative hazard function
fit = survfit(formula=coxph(Surv(conjunt$initial,conjunt$final,conjunt$event)~1),
    type="aalen")
Z.na=-log(fit$surv)
# Cox-Snell residual plot
plot(fit$time,Z.na, pch=16, xlab="cum. Cox-Snell (recurrent)", ylab="Nelson-Aalen
    cumulative hazard", main=)
lines(fit$time, fit$time, col='red', lwd=1.5)
```

Appendix E. Code to reproduce the results of the case study presented in Section 5

```
# loading libraries
require(numDeriv)
require(survival)
# load bowel motility data
dat <- read.table('Bowel-Motility.csv', header=T, sep=';')
u1 = dat$id
u1 = unique(u1)
gaps = lapply(1:length(u1), function(x) dat$intTime[dat$id == u1[x]]/60)  # time =
    hours
censor = lapply(1:length(u1), function(x) dat$status[dat$id == u1[x]])
times <- list()
for(w in 1:length(gaps)) {
        times[[w]] <- 0
}
for(s in 1:length(gaps)) {
        for(i in 1:(length(gaps[[s]]))){
                times[[s]][i+1] <- times[[s]][i]+gaps[[s]][i]
    }
}
# fit of complete model - PEre model
otimPE <- optim(par=c(-0.6,1), method="BFGS", fn=fr, hessian=TRUE, cens=censor,
    temp=times, n=length(times), control=list(reltol=1e-5))
# ML estimates
estimates <- exp(otimPE$par)
# calculate the information matrix
FisherInf <- hess(exp(otimPE$par), cens=censor, temp=times, n=length(times))
aux <- solve(FisherInf)
# variance and covariance from information matrix
vetvars <- diag(aux); covars <- aux[1,2]
# confidence intervals for parameters
IC <- matrix(c(exp(otimPE$par)-1.96*sqrt(vetvars), exp(otimPE$par)+1.96*sqrt(
    vetvars)), ncol=2, byrow=F)
# log-likelihood function
logLikePE <- -fr(otimPE$par, cens=censor, temp=times, n=length(times))
# show results (Table 5 - PEre Model)
estimates
IC
```

```
logLikePE

# fit of submodel - HPP model
otimHPP <- optim(par=-0.3, method="BFGS", fn=fr, hessian=TRUE, cens=censor, w=gaps
    , n=length(gaps), h0=T, control=list(reltol=1e-5))
# ML estimates
estimate <- exp(otimHPP$par)
# calculate the information matrix for the HPP model
FisherHPP <- hessHPP(exp(otimHPP$par), cens=censor, n=length(censor))
vetvars1 = solve(FisherHPP)
IC1 <- c(exp(otimHPP$par)-1.96*sqrt(vetvars1), exp(otimHPP$par)+1.96*sqrt(vetvars1
    ))
# log-likelihood function
logLikeHPP <- -fr(otimHPP$par, cens=censor, w=gaps, n=length(gaps), h0=T)
# show results (Table 5 - HPP Model)
estimate
IC1
logLikeHPP

# Likelihood ratio test
LRS <- 2*(logLikePE-logLikeHPP)
p.valor <- pchisq(LRS, df=1, lower.tail = FALSE)
# Score test
alp = estimate
TS <- Test.score(param=c(alp,0.001), temp=times, cens=censor, n=length(times))
p.valor <- pchisq(TS, df=1, lower.tail = FALSE)

# Cox-Snell residuals
# Note: not run the residual for PEre model and residual for HPP model
    simultaneously!

# partial Cox-Snell residuals - PEre model
a0 = exp(otimPE$par[1]); t0 = exp(otimPE$par[2])
cs = lapply(1:length(times),function(h){tam=length(times[[h]]); res<-log((1-exp(-
    t0*exp(-a0*times[[h]][1:(tam-1)])))/(1-exp(-t0*exp(-a0*times[[h]][2:tam]))))})

# partial Cox-Snell residuals - HPP model
a01 = exp(otimHPP$par)
cs = lapply(1:length(gaps), function(h) {res1 <- a01*gaps[[h]]})

# cumulative Cox-Snell residual for each recurrence
cumcs = list()
for (k in 1:length(cs)) {
        cumcs[[k]] = cumsum(cs[[k]])
}
cumcs2 = unlist(cumcs)
# variation for residuals with same value
taman1 = length(cumcs2); taman2 = length(unique(cumcs2))
while (taman1 != taman2) {
        add = table(cumcs2)
        add1 = rownames(add[add > 1])

        for (y in 1:length(add1)) {
                count = 0
                for (x in 1:length(cumcs2)) {
                        if (add1[y] == cumcs2[x]) {
                                count = count + 1
                                if (count != 1) {
                                        cumcs2[x] = cumcs2[x] + rnorm(1,0,0.0001)
                                }
                        }
                }
        }
        taman1 = length(cumcs2); taman2 = length(unique(cumcs2))
}
event = unlist(censuras)
tam1 = unlist(lapply(cumcs,length))
```

```
id = rep(seq(1,length(tam1)),tam1)

# organize data to calculate the Nelson-Aalen estimate of the cumulative hazard
    function
conjunt = data.frame(id,event,cumcs2)
z=rep(0,length(conjunt[,1]))
conjunt=cbind(conjunt,z,z)
colnames(conjunt)=c("id","event","cumcs2","initial","final")
v = unique(conjunt$id)
tamv = length(v)
for (w in 1:tamv) {
        a = c(0,conjunt[conjunt$id==v[w],]$cumcs2)
        for (j in 1:(length(a)-1)) {
                conjunt[conjunt$id==v[w],][j,]$initial = a[j]
                conjunt[conjunt$id==v[w],][j,]$final = a[j+1]
        }
}
# Figure 3:
# Nelson-Aalen estimate of the cumulative hazard function
fit = survfit(formula=coxph(Surv(conjunt$initial,conjunt$final,conjunt$event)~1),
    type="aalen")
Z.na=-log(fit$surv)
# Cox-Snell residual plot
plot(fit$time,Z.na, pch=16, xlim=c(0,max(Z.na)), xlab="cum. Cox-Snell (recurrent)"
    , ylab="Nelson-Aalen cumulative hazard", main="")
lines(Z.na, Z.na, col='red', lwd=1.5)
```