The generalized odd log-logistic-G regression with interval-censored survival data.

Valdemiro P. Vigas [a] , Edwin M. M. Ortega [b], Gauss M. Cordeiro [c], Adriano K. Suzuki [d] and Paulo C. dos Santos Junior [e]

[a]Institute of Mathematics, Federal University of Mato Grosso do Sul, Campo Grande, MS, Brazil;
[b] Department of Exact Sciences, University of São Paulo, Piracicaba, SP, Brazi;
[c]Department of Statistics, Federal University of Pernambuco, Recife, PE, Brazil;
[d]Department of Applied Mathematics and Statistics, University of São Paulo, São Carlos, SP, Brazil;
[e]Faculty of Statistics, Federal University of Pará, Belém, PA, Brazil.

ARTICLE HISTORY
Compiled May 18, 2023

Supplemental Material

- Program developed in R for the plot of the hazard function of the GOLL-WEIBULL distribution (Figure 1a).

```
######################
#HAZARD GOLL-WEIBULL #
######################

GOLL_W<- function(tempo,alpha,theta,a,b){
gt5=(a*tempo^(a-1)/b^a)*(exp(-(tempo/b)^a))
Gt5=1-exp(-(tempo/b)^a)
ft5=(alpha*theta*gt5*(Gt5^(alpha*theta-1))*((1-Gt5^(theta))^(alpha-1)))/((Gt5^(alpha*theta)+(1-Gt5^theta)^alpha)^2)
Ft5=(Gt5^(alpha*theta))/(Gt5^(alpha*theta)+ (1-Gt5^(theta))^alpha)
St5=1-Ft5
ht5=ft5/St5
return(ht5)
}

tempo=c(seq(0,10,0.01))

v_lty <- c('longdash', 'dotdash', 'dotted', 'dashed', 'solid')


#---------------------------------
#Loading R packages
#---------------------------------
library(ggplot2)


#########################################################

ggplot(NULL, aes(x=x, colour = PDF))+
```

CONTACT Valdemiro P. Vigas. Email: valdemiro.vigas@ufms.br

```
 stat_function(fun=GOLL_W,
                data = data.frame(x = c(seq(0,10,0.1)), PDF = factor(1)),
               size=1, #red
                args = list(alpha=1,theta=1,a=1,b=5
                            )) +
   stat_function(fun=GOLL_W,
                 data = data.frame(x = c(seq(0,10,0.1)),
                                   PDF = factor(2)),
                 size=1, #blue
                 args = list(alpha=0.3,theta=0.5,a=0.5,b=2.0
                            ))+

   stat_function(fun=GOLL_W,
                data = data.frame(x = c(seq(0,10,0.1)),
                                   PDF = factor(3)),
                 size=1, #yelow
                args = list(alpha=0.7,theta=2,a=0.5,b=0.5
                            ))+

   stat_function(fun=GOLL_W,
                 data = data.frame(x = c(seq(0,10,0.1)),
                                   PDF = factor(4)),
                 size=1, #green
                args = list(alpha=0.25,theta=2,a=3,b=4.0
                            ))+

  stat_function(fun=GOLL_W,
                 data = data.frame(x = c(seq(0,10,0.1)),
                                    PDF = factor(5)),
                linetype = 1, size=1, #black
                args = list(alpha=0.25,theta=2,a=2,b=3.0
                            ))+

  scale_colour_manual(values = c("red","blue","yellow","green","black"),
 labels = c(expression(list(alpha==1,theta==1,a==1,b==5),
list(alpha==0.3,theta==0.5,a==0.5,b==2.0),
list(alpha==0.7,theta==2,a==0.5,b==0.5),
list(alpha==0.25,theta==2,a==3,b==4.0),
list(alpha==0.25,theta==2,a==2,b==3.0)
))) +
  scale_linetype_manual(values = v_lty)+

   xlab("t") +
   ylab("h(t)") +
  xlim(0, 10)+ylim(0.0, 0.8)+
#  ggtitle(expression(list(alpha==0.25,theta==60,b==3.0))) +
 theme(plot.title = element_text(hjust = 0.5)) +
  # theme_bw() +
  theme(legend.title = element_blank(),
        legend.text = element_text(size = 12),
         legend.position = c(.41, 0.83),
         plot.title = element_text(hjust = 0.5))
```

- Program developed in R for to generate the values GOLL-W model with interval-censored survival data with the MLE's for n=200 and 30% censoring percentage (Table 2).

```
rm(list=ls(all=TRUE))

#--- Funcao principal:

parallel.modelo<- function(n.size){


  sim.ICdata <- function(n=n, alpha.par=alpha.par, theta.par=theta.par, a.par=a.par, b.par=b.par,
                         noCens=FALSE,lambda.parc=lambda.parc){

    u <- runif(n,0,1)
    v <- ((1/(((1/u)-1)^(1/alpha.par)+1))^(1/theta.par))
    t <- qweibull(v, shape=a.par, scale=b.par)
    if( noCens == TRUE ){
      c <- rep(999999, n)
      tempo <- ifelse(t<= c, t, c)
    }else{
      c <- rexp(n, rate = lambda.parc)
      tempo <- ifelse(t<= c, t, c)
    }

    #-- Observed times:

    delta <- ifelse(t<= c, 1, 0)

    L <- R <- tempo * NA
    for (i in 1:n) { # para as censuras
      if (delta[i] == 0){
        L[i] <- tempo[i]
        R[i] <- Inf
      }
      else { # para as falhas
        L[i] <- 0
        add <- runif(1, 0.1, 0.5)
        R[i] <- add
        check <- (L[i] <= tempo[i] & tempo[i] < R[i])
        while (!check) {
          L[i] <- L[i] + add
          add <- runif(1, 0.1, 0.5)
          R[i] <- R[i] + add
          check <- (L[i] <= tempo[i] & tempo[i] < R[i])
        }
      }
    }

    dados <- data.frame(L, R, tempo, delta)

    return(dados)
  }

  StGOLLweibull=function(tempo=tempo, alpha.par=alpha.par, theta.par=theta.par, a.par=a.par, b.par=b.par){


    gt1=(a.par*tempo^(a.par-1)/b.par^a.par)*(exp(-(tempo/b.par)^a.par))
    Gt1=(1-exp(-(tempo/b.par)^a.par))
    ft1=(alpha.par*theta.par*gt1*(Gt1^(alpha.par*theta.par-1)) *(1-Gt1^theta.par)^(alpha.par-1)) /
      (((Gt1^(alpha.par*theta.par)) + (1-Gt1^theta.par)^(alpha.par))^2)
    Ft1=(Gt1^(alpha.par*theta.par))/(((Gt1^(alpha.par*theta.par))+(1-Gt1^theta.par)^alpha.par))
    St1=1-Ft1

    return(St1)
  }
```

```
#log likelihood:

logWeibull_GOLLL_IC <- function(p=p, l=l, r=r){



  r <- ifelse(is.infinite(r), 9999, r)
  p2 <- StGOLLweibull(tempo=l, alpha.par=p[1], theta.par=p[2], a.par=p[3], b.par=p[4])
  p1 <- StGOLLweibull(tempo=r, alpha.par=p[1], theta.par=p[2], a.par=p[3], b.par=p[4])
  lik <- p2-p1

  return(sum(log(lik)))
  #return(-sum(log(lik)))

}

ic.assintotico <- function(emv, ep){
  return(emv + c(-1,1)*qnorm(1-0.05/2)*ep)
}

#--- Carregando alguns pacotes:

require(survival)
require(gamlss.dist)
require(eha)
require(MASS)
require(numDeriv)

#--- Fixando a semente de dados:

set.seed(11)

#--- Definindo valores para o MC:

simula <- 1000   # Numero de simulacoes
n      <- n.size # Tamanho amostral

#--- Parametros
alpha=1
theta=2.5
a=3
b=4.5


lambdac <- 0.065 # 70% de falha 30% censura


#--- Definindo vetores para armazenamento

param <- c(alpha, theta, a, b)
event.prop <- rep(NA, simula)
est <- matrix(nrow=simula, ncol=length(param))
ep <- matrix(nrow=simula, ncol=length(param))
bias <- matrix(nrow=simula, ncol=length(param))
pc.par <- rep(0, length(param))

cor.times <- rep(NA, simula)
erro.max <- 0
conv <- rep(NA, simula)

L     <- c()
R     <- c()
Delta <- c()

#psimula <- tkProgressBar(title = "Simulação - MC", min = 0, max = simula, width = 300)
#time.simula <- system.time(
```

```
for(s in 1:simula){

  repeat{
    #
    # Gerando os dados:

    dadosIC <- sim.ICdata(n=n.size, alpha.par=alpha, theta.par=theta, a.par=a, b.par=b,
                          noCens = FALSE,lambda.parc=lambdac)

    #colocando os valores iniciais p/ os parametros:

    ini.info <- param



    test <- try(optim(par = ini.info, fn=logWeibull_GOLLL_IC, gr = NULL,
                      method = "Nelder-Mead",control=list(fnscale=-1),
                      hessian = TRUE, l=dadosIC$L, r=dadosIC$R),
                silent = TRUE)

    if(class(test)!="try.error"){

      event.prop[s] <- sum(dadosIC$delta)/n

      #Otimizando a função sem as derivadas acima:
      maxloglik <-optim(par = ini.info, fn=logWeibull_GOLLL_IC, gr = NULL,
                        method = "Nelder-Mead",control=list(fnscale=-1),
                        hessian = TRUE, l=dadosIC$L, r=dadosIC$R)
      hess <- maxloglik$hessian

      var.cov <- solve(-hess)

      if(sum(diag(var.cov)<0)>0){
        while( sum(diag(var.cov)<0)>0 ){

          dadosIC <- sim.ICdata(n=n.size, alpha.par=alpha, theta.par=theta, a.par=a, b.par=b,
                                noCens = FALSE,lambda.parc=lambdac)

          maxloglik <- optim(par = ini.info, fn=logWeibull_GOLLL_IC, gr = NULL,
                             method = "BFGS",control=list(fnscale=-1),
                             hessian = TRUE, l=dadosIC$L, r=dadosIC$R)

          hess <- maxloglik$hessian
          var.cov <- solve(-hess)
        }
      }

      # EMV

      est[s,] <- maxloglik$par

      # EP:

      ep[s,] <- sqrt(diag(var.cov))

      # Vi?s: IC = EMV +- 1,96 * EP

      bias[s,] <- est[s,]-param


      # Calculando a probilidade de cobertura:

      #--- Falha:

      for( k in 1:length(param)){
        ic.par <- ic.assintotico(est[s,k], ep[s,k])
```

```
          if( (param[k]>= ic.par[1]) & (param[k]<= ic.par[2]) )
          {
            pc.par[k] <- pc.par[k] +1
          }

        }

        # Convergencia:

        #conv[s] <- maxloglik$code

        break
      }else{
        erro.max <- erro.max+1
      }

    }


    # Sys.sleep(0.1)
    # setTkProgressBar(psimula, s, label=paste("Iter:", s,"-",round(s/simula*100, 0), "% done", "N=", n.size))
    #

  }
  #)


  tabela <- cbind(est=apply(est, 2, mean),
                  vies=apply(bias, 2, mean),
                  reqm=sqrt( ((apply(est,2, mean)-param)^2) +  apply(est,2, var) ),
                  ep= apply(ep, 2, mean),
                  pc=pc.par/simula)
  rownames(tabela) <- c("alpha", "theta", "a", "b")


  write.table(tabela, paste("tabela_",n,".txt", sep=""), sep=" ")
  write.table(est, paste("est_",n,".txt", sep=""), sep=" ")
  write.table(erro.max, paste("erro_",n,".txt", sep=""), sep=" ")
  write.table(mean(event.prop), paste("propfalha_",n,".txt", sep=""), sep=" ")
}

#
parallel.modelo(n.size = 200)
```

- Scripts to obtain the estimates of the parameters for the GOLL-LL regression using the classical and Bayesian methods (Table 12).

```
####=> Frequentist method


aux.lvero=numeric()
n=length(delta)
value=c(-0.5987634,  0.7000517,  1.1965284 , 1.3219336,  4.1120858 , 1.1114253,  9.3583004)

lvero = function(x) {
  beta0 = x[1]
  beta1 = x[2]
  beta2 = x[3]
  beta3 = x[4]
  gamma = x[5]
  alpha<-x[6]
  theta<-x[7]

    beta = exp(beta0+beta1*Low+beta2*Medium+beta3*High)

  gl=(beta/gamma)*((yl/gamma)^(beta-1))*((1+(yl/gamma)^(beta))^(-2))
  Gl=(1+(yl/gamma)^(-beta))^(-1)
  fl=(alpha*(theta)*gl*(Gl^(alpha*(theta)-1))*(1-Gl^(theta))^(alpha-1))/
  (((Gl^(alpha*(theta)))+(1-Gl^(theta))^(alpha))^2)
  Fl=(Gl^(alpha*(theta)))/(((Gl^(alpha*(theta)))+(1-Gl^(theta))^alpha))
  Sl=(1-Fl)
  gr=(beta/gamma)*((yr/gamma)^(beta-1))*((1+(yr/gamma)^(beta))^(-2))
  Gr=(1+(yr/gamma)^(-beta))^(-1)
  #
  fr=(alpha*(theta)*gr*(Gr^(alpha*(theta)-1))*(1-Gr^(theta))^(alpha-1))/
  (((Gr^(alpha*(theta)))+(1-Gr^(theta))^(alpha))^2)
  Fr=(Gr^(alpha*(theta)))/(((Gr^(alpha*(theta)))+(1-Gr^(theta))^alpha))
  Sr=(1-Fr)
  m1   = Sl - Sr
  m2   = Sl
  aux.lvero = delta*(log(m1))+(1-delta)*(log(m2))
  -sum(aux.lvero)
}

esttimate= optim(value,lvero,method="BFGS",hessian=TRUE)

####=> Bayesian method

set.seed(2993)

library(rjags)
library(R2jags)
library(coda)
library(mcmcplots)
require(rjags)
require(R2jags)
require(coda)
require(mcmcplots)
require(LaplacesDemon)

n = length(delta)
cat( "
      data{
      C <- 1000
      for(i in 1:n){zeros[i] <- 0}
      }
     model{
      for(i in 1:n){

  #tl
log(beta[i])=(b0+b1*Low[i]+b2*Medium[i]+b3*High[i])
gl[i]=(beta[i]/gamma)*((yl[i]/gamma)^(beta[i]-1))*((1+(yl[i]/gamma)^(beta[i]))^(-2))
```

```
Gl[i]=1-(1/(1+(yl[i]/gamma)^beta[i]))
  fl[i]=(alpha*(theta)*gl[i]*(Gl[i]^(alpha*(theta)-1))*(1-Gl[i]^(theta))^(alpha-1))/

  (((Gl[i]^(alpha*(theta))) + (1-Gl[i]^(theta))^(alpha))^2)
  Fl[i]=(Gl[i]^(alpha*(theta)))/(((Gl[i]^(alpha*(theta)))+(1-Gl[i]^(theta))^alpha))
  Sl[i]=(1-Fl[i])


  #tr
gr[i]=(beta[i]/gamma)*((yr[i]/gamma)^(beta[i]-1))*((1+(yr[i]/gamma)^(beta[i]))^(-2))
Gr[i]=1-(1/(1+(yr[i]/gamma)^beta[i]))
  fr[i]=(alpha*(theta)*gr[i]*(Gr[i]^(alpha*(theta)-1))*(1-Gr[i]^(theta))^(alpha-1))/

  (((Gr[i]^(alpha*(theta))) + (1-Gr[i]^(theta))^(alpha))^2)
  Fr[i]=(Gr[i]^(alpha*(theta)))/(((Gr[i]^(alpha*(theta)))+(1-Gr[i]^(theta))^alpha))
  Sr[i]=(1-Fr[i])
#
  m1[i]=Sl[i]-Sr[i]
  m2[i]=Sl[i]
ll[i]<-delta[i]*(log(m1[i]))+(1-delta[i])*(log(m2[i]))
zeros.mean[i] <- -ll[i] + C
zeros[i] ~ dpois(zeros.mean[i])
}

#Prior's distribuition
b0~dnorm(0,0.01)
b1~dnorm(0,0.01)
b2~dnorm(0,0.01)
b3~dnorm(0,0.01)
gamma~dgamma(0.01,0.01)
alpha~dgamma(0.01,0.01)
theta~dgamma(0.01,0.01)

 }", file="modelo.jags" )
initialization <- function() {
  list("gamma"= 4, "alpha"=1, "theta"= 9,"b0"= 1,"b1"= 0.7001 ,"b2"= 1.1965 ,"b3"= -1)   }
a.dat <- list("n","yl","yr","delta","Low","Medium","High")
parameters0 <-c("gamma","alpha","theta","b0","b1","b2","b3")
model11<- jags(data = a.dat,
               inits = initialization,
               parameters.to.save = parameters0,
               n.chains = 2,
               n.iter = 35000,
               n.burnin = 5000,
               n.thin=20,
               model.file ='modelo.jags')
print(model11)
```

- Scripts to obtain the QQ-plots for the deviance residuals and Summary of the SW statistic in the GOLL-LL regression with 0% censoring percentages (Figure 3 and Table 6).

```
#--- Limpando o R:

rm(list=ls(all=TRUE))


parallel.modelo<- function(n.size){



  sim.ICdata <- function(n=n, alpha.par=alpha.par, theta.par=theta.par, a.par=a.par, beta.par=beta.par,
                         noCens=FALSE,lambda.parc=lambda.parc){


    x1 <- rbinom(n, size = 1, prob = 0.5)
    X <- cbind(1, x1)
    b.par <- as.numeric(exp(crossprod(t(X),beta.par)))
    u <- runif(n,0,1)
    v <- ((1/(((1/u)-1)^(1/alpha.par)+1))^(1/theta.par))
    #  t <- qweibull(v, shape=b.par, scale=a.par)
    t=qllogis(v, shape=b.par, scale=a.par)

    if( noCens == TRUE ){
      c <- rep(999999, n)
      tempo <- ifelse(t<= c, t, c)
    }else{
      c <- rexp(n, rate = lambda.parc)
      tempo <- ifelse(t<= c, t, c)
    }

    #-- Observed times:

    delta <- ifelse(t<= c, 1, 0)

    L <- R <- tempo * NA
    for (i in 1:n) { # para as censuras
      if (delta[i] == 0){
        L[i] <- tempo[i]
        R[i] <- Inf
      }
      else { # para as falhas
        L[i] <- 0
        add <- runif(1, 0.1, 0.5)
        R[i] <- add
        check <- (L[i] <= tempo[i] & tempo[i] < R[i])
        while (!check) {
          L[i] <- L[i] + add
          add <- runif(1, 0.1, 0.5)
          R[i] <- R[i] + add
          check <- (L[i] <= tempo[i] & tempo[i] < R[i])
        }
      }
    }

    dados <- data.frame(L, R, tempo, delta, x1)

    return(dados)
  }

  StGOLLLlogistic=function(tempo=tempo, alpha.par=alpha.par, theta.par=theta.par,
                           a.par=a.par, beta.par=beta.par, X.cov=X.cov){
```

```
    X <- cbind(1, X.cov)
    b.par <- as.numeric(exp(crossprod(t(X),beta.par)))

    gt1=(b.par/a.par)*((tempo/a.par)^(b.par-1))*((1+(tempo/a.par)^(b.par))^(-2))
    Gt1=(1+(tempo/a.par)^(-b.par))^(-1)
    ft1=(alpha.par*theta.par*gt1*(Gt1^(alpha.par*theta.par-1)) *(1-Gt1^theta.par)^(alpha.par-1)) /
      (((Gt1^(alpha.par*theta.par)) + (1-Gt1^theta.par)^(alpha.par))^2)
    Ft1=(Gt1^(alpha.par*theta.par))/(((Gt1^(alpha.par*theta.par))+(1-Gt1^theta.par)^alpha.par))
    St1=1-Ft1
    return(St1)
  }


  #log likelihood:

  logLlogistic_GOLLL_IC <- function(p=p, l=l, r=r, X.cov=X.cov){

    #p <- ini.info
    #npar <- length(p)
    #l <- dadosIC$L
    #r <- dadosIC$R

    r <- ifelse(is.infinite(r), 9999, r)
    p2 <- StGOLLLlogistic(tempo=l, alpha.par=p[1], theta.par=p[2], a.par=p[3], beta.par=c(p[4],p[5]), X.cov = X.cov)
    p1 <- StGOLLLlogistic(tempo=r, alpha.par=p[1], theta.par=p[2], a.par=p[3], beta.par=c(p[4],p[5]), X.cov = X.cov)
    lik <- p2-p1

    return(sum(log(lik)))
    #return(-sum(log(lik)))

  }



  ic.assintotico <- function(emv, ep){
    return(emv + c(-1,1)*qnorm(1-0.05/2)*ep)
  }



  require(survival)
  #require(xtable)
  #require(nlme)
  require(gamlss.dist)
  require(eha)
  require(MASS)
  require(numDeriv)

  #--- Fixando a semente de dados:

set.seed(22)

  #--- Definindo valores para o MC:

  simula <- 400# Numero de simulacoes
  n      <- n.size # Tamanho amostral

  #--- Parametros GOLL_LL:

  a=2.5
  beta.reg=c(-0.6,0.8)
  alpha=3.5
  theta=1.0

lambdac <- 0.0009#  0%# censura
```

```
# censura


 #--- Definindo vetores para armazenamento

 param <- c(alpha, theta, a, beta.reg)
 event.prop <- rep(NA, simula)
 est <- matrix(nrow=simula, ncol=length(param))
 bias <- matrix(nrow=simula, ncol=length(param))
 cor.times <- rep(NA, simula)
 erro.max <- 0
 conv <- rep(NA, simula)

 L     <- c()
 R     <- c()
 Delta <- c()
 X1    <- c()

 #psimula <- tkProgressBar(title = "Simulação - MC", min = 0, max = simula, width = 300)
 #time.simula <- system.time(
 for(s in 1:simula){

   repeat{
     # #
     # Gerando os dados:

     dadosIC <- sim.ICdata(n=n.size, alpha.par=alpha, theta.par=theta, a.par=a, beta.par=beta.reg,
                           noCens = FALSE,lambda.parc=lambdac)

     #colocando os valores iniciais p/ os par?metros:

     ini.info <- param

     # mle <- optim(par = ini.info, fn=logLlogistic_GOLLL_IC, gr = NULL,
     #                           method = "BFGS",control=list(fnscale=-1), hessian = FALSE,
     # l=dadosIC$L, r=dadosIC$R, X.cov=dadosIC$x1)
     # mle$par
     # param

     test <- try(optim(par = ini.info, fn=logLlogistic_GOLLL_IC, gr = NULL,
                       method = "Nelder-Mead",control=list(fnscale=-1),
                       hessian = FALSE, l=dadosIC$L, r=dadosIC$R, X.cov=dadosIC$x1), silent = TRUE)

     if(class(test)!="try.error"){
       #      write.table(dadosIC, paste("dadosIC_",n,"_",s,".txt"), sep=" ")
       L     <- cbind(L, dadosIC$L)
       R     <- cbind(R, dadosIC$R)
       Delta <- cbind(Delta, dadosIC$delta)
       X1 <- cbind(X1, dadosIC$x1)

       event.prop[s] <- sum(dadosIC$delta)/n

       #Otimizando a fun??o sem as derivadas acima:
       maxloglik <- optim(par = ini.info, fn=logLlogistic_GOLLL_IC, gr = NULL,
                          method = "Nelder-Mead",control=list(fnscale=-1), hessian = FALSE,
                          l=dadosIC$L, r=dadosIC$R,X.cov=dadosIC$x1)

       est[s,] <- maxloglik$par

       # Vi?s:

       bias[s,] <- est[s,]-param

       # Convergencia:

       #conv[s] <- maxloglik$code
```

```
        break
      }else{
        erro.max <- erro.max+1
      }

    }


    # Sys.sleep(0.1)
    # setTkProgressBar(psimula, s, label=paste("Iter:", s,"-",round(s/simula*100, 0), "% done", "N=", n.size))
    #

  }
  #)


  tabela <- cbind(est=apply(est, 2, mean), vies=apply(bias, 2, mean),reqm=sqrt( ((apply(est,2, mean)-param)^2)
  +  apply(est,2, var) ))
  rownames(tabela) <- c("alpha", "theta", "a", "b0", "b1")
  #tabela


  write.table(tabela, paste("tabela_",n,".txt", sep=""), sep=" ")
  write.table(est, paste("est_",n,".txt", sep=""), sep=" ")
  write.table(erro.max, paste("erro_",n,".txt", sep=""), sep=" ")
  write.table(mean(event.prop), paste("propfalha_",n,".txt", sep=""), sep=" ")
  write.table(L, paste("L_",n,".txt", sep=""), sep=" ")
  write.table(R, paste("R_",n,".txt", sep=""), sep=" ")
  write.table(Delta, paste("Delta_",n,".txt", sep=""), sep=" ")
  write.table(X1, paste("X1_",n,".txt", sep=""), sep=" ")

}


#

parallel.modelo(n.size = 200)

################################################

install.packages("dplyr")
require(dplyr)
#chamar o diretorio em "meus documentos"

n=200
B=400

dadosL1=read.table("L_200.txt",head=TRUE)
attach(dadosL1)
tw=select(dadosL1,starts_with("V"))
tw1=c(tw)
tL2=as.numeric(unlist(tw1))
yl=matrix(tL2,n,B)



#
dadosR1=read.table("R_200.txt",head=TRUE)
attach(dadosR1)
tr=select(dadosR1,starts_with("V"))
tr1=c(tr)
tr2=as.numeric(unlist(tr1))
yr=matrix(tr2,n,B)



#
dadosdelta=read.table("Delta_200.txt",head=TRUE)
```

```
attach(dadosdelta)
trdelta=select(dadosdelta,starts_with("V"))
tdelta1=c(trdelta)
trdelta=as.numeric(unlist(tdelta1))
delta=matrix(trdelta,n,B)

#
dadoscova=read.table("X1_200.txt",head=TRUE)
attach(dadoscova)
trcova=select(dadoscova,starts_with("V"))
tcova1=c(trcova)
tcova2=as.numeric(unlist(tcova1))
cova=matrix(tcova2,n,B)



########################################
########  GOLL-LL              ########
########################################


set.seed(21)




b0=-0.6
beta0=-0.6

b1=0.8
beta1=0.8

a=2.5
alpha=3.5
theta=1


#
b0.est=numeric()
b0.est=matrix(nrow=1,ncol=B)
#
b1.est=numeric()
b1.est=matrix(nrow=1,ncol=B)
#
alpha.est=numeric()
alpha.est=matrix(nrow=1,ncol=B)
#
theta.est=numeric()
theta.est=matrix(nrow=1,ncol=B)
#
a.est=numeric()
a.est=matrix(nrow=1,ncol=B)
#
b11=numeric()
b11=matrix(nrow=n,ncol=B)
#
gl=numeric()
gl=matrix(nrow=n,ncol=B)
Gl=numeric()
Gl=matrix(nrow=n,ncol=B)
fl=numeric()
fl=matrix(nrow=n,ncol=B)
Fl=numeric()
Fl=matrix(nrow=n,ncol=B)
Sl=numeric()
Sl=matrix(nrow=n,ncol=B)
#
gr=numeric()
```

```
gr=matrix(nrow=n,ncol=B)
Gr=numeric()
Gr=matrix(nrow=n,ncol=B)
fr=numeric()
fr=matrix(nrow=n,ncol=B)
Fr=numeric()
Fr=matrix(nrow=n,ncol=B)
Sr=numeric()
Sr=matrix(nrow=n,ncol=B)
#
m1=numeric()
m1=matrix(nrow=n,ncol=B)
m2=numeric()
m2=matrix(nrow=n,ncol=B)
#
max_vero=numeric()
max_vero=matrix(nrow=n,ncol=B)
log_vero=numeric()
log_vero=matrix(nrow=n,ncol=B)
soma_log_vero=numeric()
soma_log_vero=matrix(nrow=1,ncol=B)
#
resu_max_vero_aleatorio=numeric()
resu_max_vero_aleatorio=matrix(nrow=1,ncol=B)
convergencia_aleatorio=numeric()
convergencia_aleatorio=matrix(nrow=1,ncol=B)

for (i in 1:B){
  programaGOLL_LL=function(x){
    #
    beta0=x[1]
    beta1=x[2]
    a1=x[3]
    alpha1=x[4]
    theta1=x[5]

    #tl
    #  gl[,i]=(a1*yl[,i]^(a1-1)/b1^a1)*(exp(-(yl[,i]/b1)^a1))
    #  Gl[,i]=(1-exp(-(yl[,i]/b1)^a1))
    b11[,i]=exp(beta0+beta1*cova[,i])
    gl[,i]=(b11[,i]/a1)*((yl[,i]/a1)^(b11[,i]-1))*((1+(yl[,i]/a1)^(b11[,i]))^(-2))
    Gl[,i]=(1+(yl[,i]/a1)^(-b11[,i]))^(-1)
    fl[,i]=(alpha1*(theta1)*gl[,i]*(Gl[,i]^(alpha1*(theta1)-1))*(1-Gl[,i]^(theta1))^(alpha1-1))/

    (((Gl[,i]^(alpha1*(theta1))) + (1-Gl[,i]^(theta1))^(alpha1))^2)
    Fl[,i]=(Gl[,i]^(alpha1*(theta1)))/(((Gl[,i]^(alpha1*(theta1)))+(1-Gl[,i]^(theta1))^alpha1))
    Sl[,i]=(1-Fl[,i])
    #tr
    #   gr[,i]=(a1*yr[,i]^(a1-1)/b1^a1)*(exp(-(yr[,i]/b1)^a1))
    #   Gr[,i]=(1-exp(-(yr[,i]/b1)^a1))

    gr[,i]=(b11[,i]/a1)*((yr[,i]/a1)^(b11[,i]-1))*((1+(yr[,i]/a1)^(b11[,i]))^(-2))
    Gr[,i]=(1+(yr[,i]/a1)^(-b11[,i]))^(-1)
    fr[,i]=(alpha1*(theta1)*gr[,i]*(Gr[,i]^(alpha1*(theta1)-1))*(1-Gr[,i]^(theta1))^(alpha1-1))/

    (((Gr[,i]^(alpha1*(theta1))) + (1-Gr[,i]^(theta1))^(alpha1))^2)
    Fr[,i]=(Gr[,i]^(alpha1*(theta1)))/(((Gr[,i]^(alpha1*(theta1)))+(1-Gr[,i]^(theta1))^alpha1))
    Sr[,i]=(1-Fr[,i])
    #
    m1[,i]=Sl[,i]-Sr[,i]
    m2[,i]=Sl[,i]
    max_vero[,i]=(m1[,i]^(delta[,i]))*(m2[,i]^(1-delta[,i]))
    log_vero[,i]=log(max_vero[,i])
    soma_log_vero[,i]=-sum(log_vero[,i])
    return(soma_log_vero[,i])
  }

  chute=c(b0,b1,a,alpha,theta)
```

```
  resultados=suppressWarnings(optim(chute,fn=programaGOLL_LL,method="Nelder-
  Mead",control=list(maxit=300000000),hessian=FALSE))
  #
  b0.est[i]=resultados$par[1]
  b1.est[i]=resultados$par[2]
  a.est[i]=resultados$par[3]
  alpha.est[i]=resultados$par[4]
  theta.est[i]=resultados$par[5]
  resu_max_vero_aleatorio[i]=programaGOLL_LL(c(resultados$par[1],resultados$par[2],resultados$par[3],

  resultados$par[4],resultados$par[5]))
  convergencia_aleatorio[i]=resultados$convergence
  }

# Encontrando os residuos

res.martingale=numeric()
res.martingale=matrix(nrow=n,ncol=B)
#
res.martingale1=numeric()
res.martingale1=matrix(nrow=n,ncol=B)
#
sinal=numeric()
sinal=matrix(nrow=n,ncol=B)

#
r_deviance1=numeric()
r_deviance1=matrix(nrow=n,ncol=B)

n=200
B=400


for (j in 1:B){
  for(i in 1:n){

    #tl
    b11[i,j]=exp(b0.est[,j]+b1.est[,j]*cova[i,j])
    gl[i,j]=(b11[i,j]/a.est[,j])*((yl[i,j]/ a.est[,j])^(b11[i,j]-1))*((1+(yl[i,j]/ a.est[,j])^(b11[i,j]))^(-2))
    Gl[i,j]=(1+(yl[i,j]/a.est[,j])^(-b11[i,j]))^(-1)
    fl[i,j]=(alpha.est[,j]*(theta.est[,j])*gl[i,j]*(Gl[i,j]^(alpha.est[,j]*(theta.est[,j])-1))*
    (1-Gl[i,j]^(theta.est[,j]))^(alpha.est[,j]-1))
    /(((Gl[i,j]^(alpha.est[,j]*(theta.est[,j]))) +
(1-Gl[i,j]^(theta.est[,j]))^(alpha.est[,j]))^2)
    Fl[i,j]=(Gl[i,j]^(alpha.est[,j]*(theta.est[,j])))/
(((Gl[i,j]^(alpha.est[,j]*(theta.est[,j])))+(1-Gl[i,j]^(theta.est[,j]))^alpha.est[,j]))
    Sl[i,j]=(1-Fl[i,j])

    #tr
    gr[i,j]=(b11[i,j]/a.est[,j])*((yr[i,j]/a.est[,j])^(b11[i,j]-1))*((1+(yr[i,j]/a.est[,j])^(b11[i,j]))^(-2))
    Gr[i,j]=(1+(yr[i,j]/a.est[,j])^(-b11[i,j]))^(-1)
 fr[i,j]=(alpha.est[,j]*(theta.est[,j])*gr[i,j]*(Gr[i,j]^(alpha.est[,j]*(theta.est[,j])-1))*
 (1-Gr[i,j]^(theta.est[,j]))^(alpha.est[,j]-1))
 /(((Gr[i,j]^(alpha.est[,j]*(theta.est[,j])))+
(1-Gr[i,j]^(theta.est[,j]))^(alpha.est[,j]))^2)
      Fr[i,j]=(Gr[i,j]^(alpha.est[,j]*(theta.est[,j])))/
      (((Gr[i,j]^(alpha.est[,j]*(theta.est[,j])))+
(1-Gr[i,j]^(theta.est[,j]))^alpha.est[,j]))
    Sr[i,j]=(1-Fr[i,j])
#
if(delta[i,j]==1){
      res.martingale1[i,j]=(Sl[i,j]*log(Sl[i,j])-Sr[i,j]*log(Sr[i,j]))/(Sl[i,j]-Sr[i,j])
      r_deviance1[i,j]<-sign(res.martingale1[i,j])*(-2*(res.martingale1[i,j]+log(1-res.martingale1[i,j])))^0.5
    }
    else{
      res.martingale1[i,j]=log(Sl[i,j])
r_deviance1[i,j]<-sign(res.martingale1[i,j])*(-2*(res.martingale1[i,j]))^0.5
    }
```

```
 }
}

# Deviance

teste <- r_deviance1
ordem_res=apply(teste,2,sort)
ordem_res


media_res=apply(ordem_res,1,mean)
media_res

qqnorm(media_res,pch=19,ylim = c(-3,4),xlim=c(-3.5,4),
  ylab="rDi",xlab="N(0,1) quantiles",main = "",cex.lab=1.2,cex.axis=1.2)
qqline(media_res, datax = F, distribution = qnorm,
       col=2,lwd=3)

# ggplot2 ------------------------------------------------------------------
library(ggplot2)
library(magrittr)
#library(plotly)
#require(plotly)


media_res1_qqnorm <- qqnorm(media_res)

data.frame(y = media_res1_qqnorm$y,
           x = media_res1_qqnorm$x) %>%
ggplot(aes(y = y, x = x)) +
 geom_point(data = NULL,
            mapping = aes(x = media_res1_qqnorm$x,
                          y = media_res1_qqnorm$y),
            col = 'red') +
  geom_abline(slope = 1.03, intercept = 0.19, size = 1) +
  theme_gray(base_size = 15) +
  labs(x = 'Percentils da N(0, 1)',
       y = 'Resíduo deviance',
       title = '')




##############################

data.frame(y = media_res1_qqnorm$y,
           x = media_res1_qqnorm$x) %>%
 ggplot(aes(y = y, x = x)) +
#  geom_abline(lwd=1,col = 'black') +
 geom_point(data = NULL,
            mapping = aes(x = media_res1_qqnorm$x,
                          y = media_res1_qqnorm$y),
            col = 'red') +
  geom_abline(slope = 1.03, intercept = 0.33, size = 1) +
  theme_gray(base_size = 15) +
  labs(x = 'N(0, 1) Quantiles ',
       y = 'Residuals',
       title = '')

##############################################




df <- data.frame(r_deviance1)
h=do.call(rbind, lapply(df, function(x) shapiro.test(x)[c("p.value")]))
```

```
h1=matrix(h,1,400)
prop <- ifelse(h1 > .05, "1", "0")
prop1=c(prop)
prop2=as.numeric(prop1)
mean(prop2)

h3=c(h1)
summary(as.numeric(h3))
```

More codes used for the HIV infection data are available upon request by emailing authors.