```
glm.mq.poisson<-
function(x,y,offs=rep(1,nrow(x)),case.weights=rep(1,nrow(x)),maxit=100,acc=
1e-04,
var.weights=rep(1,nrow(x)),weights.x=FALSE,q=0.5,k=1.6)
{
#This is the robust glm for count data by Cantoni

        glm.rob.poisson <-
function(X,y,weights.on.x=FALSE,chuber=1.345,offset=offs)
        {
# Preliminary definitions
                mytol <- .Machine$double.eps^.25
                nb <- length(y)
                assign("nb",nb)


                basepsi <- function(x)
                {
                        x*pmin(1,chuber/abs(x))
                }
                assign("basepsi",basepsi)

                basepsiprime <- function(x)
                {
                        1*(abs(x)<chuber)
                }
                assign("basepsiprime",basepsiprime)

# Initializing....
                beta.old <-
as.vector(glm(y~X-1,family=poisson,offset=log(offset))$coeff)
                Xwith <- X


                eta <- Xwith%*%beta.old
                probab <- offset*exp(eta)
                mu <- probab
                V <- mu
                deriv.mu <- offset*exp(eta)
                r.stand <- (y-mu)/sqrt(V)
                ifelse (weights.on.x,w.x <- sqrt(1-hat(X)),w.x <-
rep(1,length=nb))
                assign("w.x",w.x)
                assign("Xwith",Xwith)
                assign("y",y)

# pmin() on the argument of pbinom() is necessary, due to the
# bad behavior of pbinom() when evaluated at values greater than size
```

```
#
# pmax() is used to avoid the warning messages due to the generation of NA
# in the ifelse procedure

                g.objective <- function(beta)
                {
                        eta <- Xwith%*%beta
                        probab <- offset*exp(eta)
                        mu <- probab
                        V <- probab
                        r.stand <- (y-mu)/sqrt(V)
                        deriv.mu <- offset*exp(eta)
                        jinf <- floor(mu-chuber*sqrt(V))
                        jsup <- floor(mu+chuber*sqrt(V))

                        if(chuber==Inf)
                        {
                                esp.cond <- rep(1,nb)
                        }
                        if(chuber!=Inf)
                        {
                                esp.cond <- -chuber*ppois(jinf,mu) +
chuber*(1-ppois(jsup,mu)) +
                                mu/sqrt(V)*(ppois(jinf,mu)-ppois(jinf-1,mu)
-(ppois(jsup,mu) -

                                                  ppois(jsup-1,mu)))
                        }
                        a.const <- apply(Xwith*as.vector(1/nb/
sqrt(V)*w.x*esp.cond*deriv.mu), 2,sum)
                        apply(Xwith*as.vector(1/nb/
sqrt(V)*w.x*basepsi(r.stand)*deriv.mu),2,sum)-a.const
                }
                assign("g.objective",g.objective)

                grad.g <- function(beta)
                {
                        delta <- .Machine$double.eps^.5
                        Ident <- diag(1,length(beta))
                        1/delta*(apply(beta+delta*Ident,2,g.objective)-
as.vector(g.objective(beta)))
                }
                tmp.times<-0
# Main
                repeat
                {tmp.times<-tmp.times+1
                        g.old <- g.objective(beta.old)
                        grad.g.old <- grad.g(beta.old)
                        csi <- solve(grad.g.old,-g.old)
                        beta.new <- as.vector(beta.old+csi)
```

```
                        if(abs(max(beta.old-beta.new))/abs(max(beta.old)) <
mytol) break
                        if (tmp.times>100)break
                        beta.old <- beta.new
                        NULL
                }

                eta <- Xwith%*%beta.old
                fit <- offset*exp(eta)
                list(coef=beta.old,fitted.values=fit)
        }

#Stopping rule
        irls.delta <- function(old, new) abs(max(old-new))/abs(max(old))
        if (qr(x)$rank < ncol(x))
        stop("X matrix is singular")
        if (length(case.weights) != nrow(x))
        stop("Length of case.weights must equal number of observations")
        if (any(case.weights < 0))
        stop("Negative case.weights are not allowed")
        n<-length(case.weights)
        ifelse (weights.x,w.x <- sqrt(1-hat(x)),w.x <- rep(1,length=n))
        assign("w.x",w.x)
#We fit the glm.rob for computing the starting values

        temp.rob <-glm.rob.poisson
(X=x,y=y,weights.on.x=weights.x,chuber=k,offset=offs)
        resid.init <- y-temp.rob$fitted.values
        fit.init <- temp.rob$fitted.values
        phi.init<-1
        done <- FALSE
        conv <- NULL
        qest <- matrix(0, nrow = ncol(x), ncol = length(q))
        qfit <- matrix(0, nrow = nrow(x), ncol = length(q))
        qres <- matrix(0, nrow = nrow(x), ncol = length(q))
        qvar <- matrix(0, nrow = ncol(x), ncol = length(q))
        qphi<-NULL
        for(i in 1:length(q)) {

#We define the starting values
                resid <- resid.init
                fit<-fit.init
                phi<-phi.init
                a.j<- case.weights
                w<-case.weights
                coef<-temp.rob$coef

                for (iiter in 1:maxit) {

                        resid.old <- resid
```

```
                        coef.old<-coef


# We define the probability mu=t*exp(xb)
                        probab<-fit
                        mu <- probab
                        deriv.mu <- mu
#We define the variance
                        V <- phi*probab

#We define the scale
                        scale<- c(sqrt(V))

#We standardize the residuals
                        r.stand <- (y-mu)/sqrt(V)

#we compute i1 and i2
                        jinf <- floor(mu-k*sqrt(V))
                        jsup <- floor(mu+k*sqrt(V))


#We compute the values of a_j(b)
                        if(k==Inf)
                        {
                                a.j <- rep(1,n)
                        }
                        if(k!=Inf)
                        {
                                a.j <- (-k)*ppois(jinf,mu) + k*(1-
ppois(jsup,mu)) +
                                mu/sqrt(V)*(ppois(jinf,mu)-ppois(jinf-1,mu)
-(ppois(jsup,mu) -

                                                  ppois(jsup-1,mu)))  }

                        a.j<-2*a.j*(q[i]*(r.stand>0)+(1-q[i])*(r.stand<=0))

#we define a part of w_j
                        w<-diag(c(mu)/scale)*diag(c(w.x))

#we compute psi_q(res)
                        tmp <- psi.huber((resid)/scale,k=k) *
case.weights*((resid)/scale)
                        tmp1 <- 2 * (1 - q[i]) * tmp
                        tmp1[resid > 0] <- 2 * q[i] * tmp[resid > 0]
                        tmp <- tmp1

#we compute psi_q(r )-E(psi_q(r ))
                        A<-(tmp-a.j)
```

```
			if(k==Inf)
			{
				esp.carre.cond <- rep(1,n)
			}
			if(k!=Inf)
		{
				esp.carre.cond <-k*(ppois(jinf,mu)-
ppois(jinf-1,mu) +(ppois(jsup,mu) - ppois(jsup-1,mu)))+(mu^2/
V^(3/2))*(ppois(jinf-1,mu)-ppois(jinf-2,mu)-(ppois(jinf,mu)-
ppois(jinf-1,mu))-(ppois(jsup-1,mu) - ppois(jsup-2,mu))+(ppois(jsup,mu) -
ppois(jsup-1,mu)))+(mu/V^(3/2))*(ppois(jsup-1,mu) - ppois(jinf,mu))
			}
			b.j<-2*esp.carre.cond*(q[i]*(r.stand>0)+(1-
q[i])*(r.stand<=0))
			B<-diag(c(V*b.j))

#We estimate betas
			temp <- coef+solve(t(x)%*%w%*%B%*%x)%*%t(x)%*%w%*%A

			coef <- temp
			eta <- x%*%coef
			fit <- offs*exp(eta)
			resid <- y-fit

			convi <- irls.delta(coef.old, coef)
			conv <- c(conv, convi)
			done <- (convi <= acc)
			if (done)
			break
		}
		if (!done)
		warning(paste("MQPoisson failed to converge in", maxit,
"steps at q = ", q[i]))

# Asymptotic estimated variance of the robust estimator

		probab<-fit
		mu <- probab
		deriv.mu<-mu

#We define the variance
		V <- phi*probab
		r.stand <- (y-mu)/sqrt(V)
		scale<- c(sqrt(V))

		jinf <- floor(mu-k*sqrt(V))
		jsup <- floor(mu+k*sqrt(V))

		if(k==Inf)
		{
```

```
                        esp.cond <- rep(1,n)
                }
                else
        {
                        esp.cond <- -k*ppois(jinf,mu) + k*(1-
ppois(jsup,mu)) + mu/sqrt(V)*(ppois(jinf,mu)-ppois(jinf-1,mu) -
(ppois(jsup,mu) - ppois(jsup-1,mu)))
        }

                esp.cond<-2*esp.cond*(q[i]*(r.stand>0)+(1-
q[i])*(r.stand<=0))
                a.const <- apply(x*as.vector(1/n/
sqrt(V)*w.x*esp.cond*deriv.mu), 2,sum)


                if(k==Inf)
                {
                        esp.carre.cond <- 1
                }
                else
        {
                        esp.carre.cond <- k^2*(ppois(jinf,mu)+1-
ppois(jsup,mu))+1/V*(mu^2*(2*ppois(jinf-1,mu)-ppois(jinf-2,mu)-
ppois(jinf,mu)-2*ppois(jsup-1,mu)+ppois(jsup-2,mu)+ppois(jsup,mu))
+mu*(ppois(jsup-1,mu)-ppois(jinf-1,mu)))
        }
                esp.carre.cond<-4*esp.carre.cond*(q[i]*(r.stand>0)+(1-
q[i])*(r.stand<=0))^2
                matQaux <- as.vector(esp.carre.cond/V*w.x^2*deriv.mu^2)
                matQ1 <- (1/n)*t(x)%*%(matQaux*x)
                matQ2 <- a.const%*%t(a.const)
                matQ <- matQ1-matQ2

                if(k==Inf)
                {
                        esp.psi.score <- 1/sqrt(V)
                }
                else
        {
                        esp.psi.score <- k*(ppois(jinf,mu)-ppois(jinf-1,mu)
+(ppois(jsup,mu) - ppois(jsup-1,mu)))+(mu^2/V^(3/2))*(ppois(jinf-1,mu)-
ppois(jinf-2,mu)-(ppois(jinf,mu)-ppois(jinf-1,mu))-(ppois(jsup-1,mu) -
ppois(jsup-2,mu))+(ppois(jsup,mu) - ppois(jsup-1,mu)))+(mu/
V^(3/2))*(ppois(jsup-1,mu) - ppois(jinf,mu))
        }
                esp.psi.score<-2*esp.psi.score*(q[i]*(r.stand>0)+(1-
q[i])*(r.stand<=0))
                matMaux <- as.vector(esp.psi.score/sqrt(V)*w.x*deriv.mu^2)
                matM <- 1/n*t(x)%*%(matMaux*x)
                matMinv <- solve(matM)
```

```
		as.var <- 1/n*matMinv%*%matQ%*%matMinv



		qest[, i] <- coef
		qfit[, i] <- fit
		qres[,i] <- y-fit
		qvar[,i]<-as.numeric(round(diag(as.var),4))
	}
	list(fitted.values=qfit, var.beta=qvar,residuals=qres, q.values=q,
coefficients=qest,matQ=matQ,matM=matM)
}
```