An introduction to the GAMLSS packages in R

Fernanda De Bastiani* Cristian Villegas†

Contents

*Universidade Federal de Pernambuco, debastiani@de.ufpe.br
†Universidade de São Paulo, clobos@usp.br

1 Install and load gamlss package

```
#install.packages("gamlss")
library(gamlss)#load gamlss
#?gamlss
#?‘gamlss-package‘
#?‘gamlss.dist-package‘
#?‘gamlss.dist-package‘
#?gamlss.family
```

1.1 Basic of gamlss function

Returns an object of class “gamlss”, which is a generalized additive model for location scale and shape (GAMLSS). The function gamlss() is very similar to the gam() function in S-plus (now also in R in package gam), but can fit more distributions (not only the ones belonging to the exponential family) and can model all the parameters of the distribution as functions of the explanatory variables (e.g. using linear, non-linear, smoothing, loess and random effects terms).

This implementation of gamlss() allows modelling of up to four parameters in a distribution family, which are conventionally called mu, sigma, nu and tau.

The function gamlssNews() shows what is new in the current implementation.

```
gamlss(formula = formula(data),
       sigma.formula = ~1,
       nu.formula = ~1,
       tau.formula = ~1,
       family = NO(),
       data,
       weights = NULL,
       contrasts = NULL,
       method = RS(),
       start.from = NULL,
       mu.start = NULL,
       sigma.start = NULL,
       nu.start = NULL,
       tau.start = NULL,
       mu.fix = FALSE,
       sigma.fix = FALSE,
       nu.fix = FALSE,
       tau.fix = FALSE,
       control = gamlss.control(...),
       i.control = glim.control(...), ...)
```

Rigby, R. A. and Stasinopoulos D. M. (2005). Generalized additive models for location, scale and shape,(with discussion), Appl. Statist., 54, part 3, pp 507-554.

Rigby, R. A., Stasinopoulos, D. M., Heller, G. Z., and De Bastiani, F. (2019) Distributions for modeling location, scale, and shape: Using GAMLSS in R, Chapman and Hall/CRC. An older version can be found in https://www.gamlss.com/.

Stasinopoulos D. M. Rigby R.A. (2007) Generalized additive models for location scale and shape (GAMLSS) in R. Journal of Statistical Software, Vol. 23, Issue 7, Dec 2007, https://www.jstatsoft.org/v23/i07/.

Stasinopoulos D. M., Rigby R.A., Heller G., Voudouris V., and De Bastiani F., (2017) Flexible Regression and Smoothing: Using GAMLSS in R, Chapman and Hall/CRC.

2 Aids Cases in England and Wales (aids)

The quarterly reported AIDS cases in the U.K. from January 1983 to March 1994 obtained from the Public Health Laboratory Service, Communicable Disease Surveillance Centre, London.

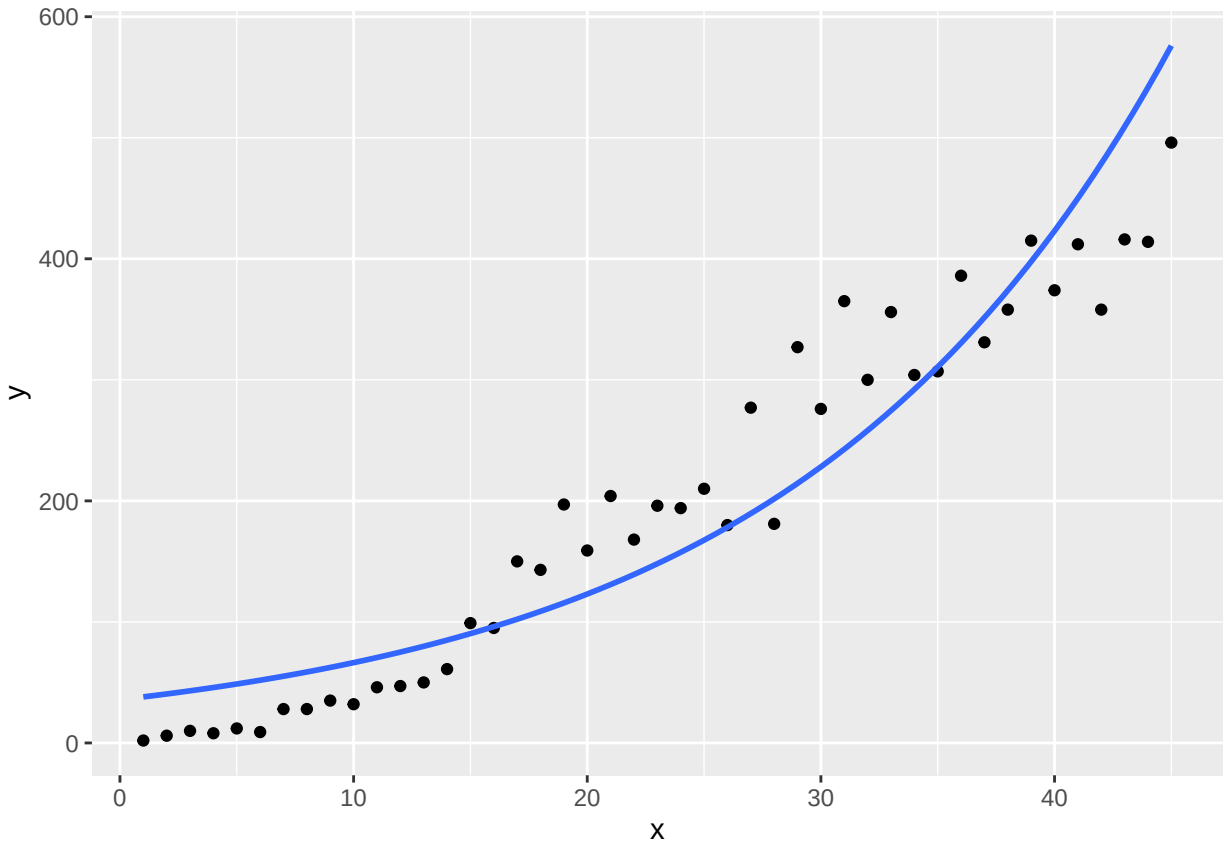
A data frame with 45 observations on the following 3 variables.

- y the number of quarterly aids cases in England and Wales: a numeric vector
- x time in months from January 1983, 1:45 : a numeric vector

Stasinopoulos, D.M. and Rigby, R. A. (1992). Detecting break points in generalized linear models. Computational Statistics and Data Analysis, 13, 461–471.

2.1 Poisson regression using P-Splines

```
# pb() and ps() functions
data(aids)
#?aids

library(ggplot2)
ggplot(aids,aes(x,y))+
  geom_point()+
  geom_smooth(method = "glm",
              formula = y~x,
              se=FALSE,
              method.args = list(family = poisson(link="log")))
```




```
#pb(x, df = NULL, lambda = NULL, max.df=NULL,control = pb.control(...), ...)
#pb(), the current version of P-splines which uses SVD in the fitting and therefore is the most reliable

PO()
```

```
##
## GAMLSS Family: PO Poisson
## Link function for mu    : log
```

```
show.link(family = "PO")
```

```
## $mu
## c("inverse", "log", "sqrt", "identity")
```

```
aids1<-gamlss(y~pb(x, df=10),
              data=aids,
              family=PO(mu.link = "log"),
              control = gamlss.control(trace = FALSE))

aids2<-gamlss(y~pb(x, df=3),
              data=aids,
              family=PO(mu.link = "log"),
              control = gamlss.control(trace = FALSE))

aids3<-gamlss(y~pb(x),
```

```
              data=aids,
              family=PO(mu.link = "log"),
              control = gamlss.control(trace = TRUE))
```

```
## GAMLSS-RS iteration 1: Global Deviance = 438.5074
## GAMLSS-RS iteration 2: Global Deviance = 438.5081
```

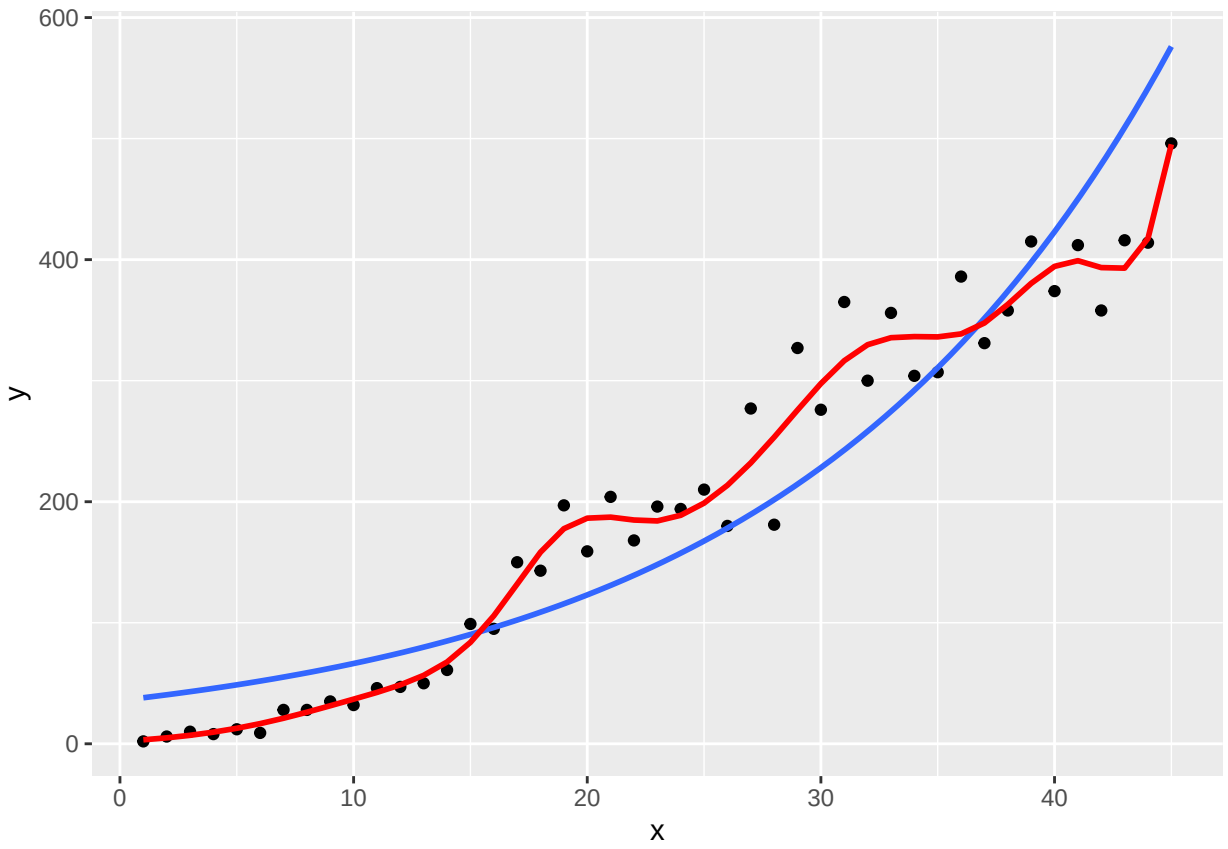
```
summary(aids1)#df=10 and edf(aids1)=12
```

```
## ******************************************************************
## Family:  c("PO", "Poisson")
##
## Call:  gamlss(formula = y ~ pb(x, df = 10), family = PO(mu.link = "log"),
##     data = aids, control = gamlss.control(trace = FALSE))
##
## Fitting method: RS()
##
## ------------------------------------------------------------------
## Mu link function:  log
## Mu Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    3.311033   0.037779   87.64   <2e-16 ***
## pb(x, df = 10) 0.070046   0.001124   62.34   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## ------------------------------------------------------------------
## NOTE: Additive smoothing terms exist in the formulas:
##  i) Std. Error for smoothers are for the linear effect only.
## ii) Std. Error for the linear terms maybe are not accurate.
## ------------------------------------------------------------------
## No. of observations in the fit:  45
## Degrees of Freedom for the fit:  12
##       Residual Deg. of Freedom:  33
##                       at cycle:  2
##
## Global Deviance:     406.393
##             AIC:     430.393
##             SBC:     452.073
## ******************************************************************
```

```
#summary(aids2)#df=3 and edf(aids2)=4.999993
#summary(aids3)#edf(aids3)=6.450721
```

```
aids$predicted <- aids1$mu.fv

ggplot(aids,aes(x,y))+
  geom_point()+
  geom_smooth(method = "glm",
              formula = y~x,
              se=FALSE,
              method.args = list(family = poisson(link="log")))+
```

```
geom_smooth(data=aids,aes(y=predicted),
            col="red",
            stat="identity")
```




```
## Some details about the gamlss output for aids dataset
names(aids1)
```

```
##  [1] "family"           "parameters"       "call"             "y"
##  [5] "control"          "weights"          "G.deviance"       "N"
##  [9] "rqres"            "iter"             "type"             "method"
## [13] "contrasts"        "converged"        "residuals"        "noObs"
## [17] "mu.fv"            "mu.lp"            "mu.wv"            "mu.wt"
## [21] "mu.link"          "mu.terms"         "mu.x"             "mu.qr"
## [25] "mu.coefficients" "mu.offset"        "mu.xlevels"       "mu.formula"
## [29] "mu.df"            "mu.nl.df"         "mu.s"             "mu.var"
## [33] "mu.coefSmo"       "mu.lambda"        "mu.pen"           "df.fit"
## [37] "pen"              "df.residual"      "P.deviance"       "aic"
## [41] "sbc"
```

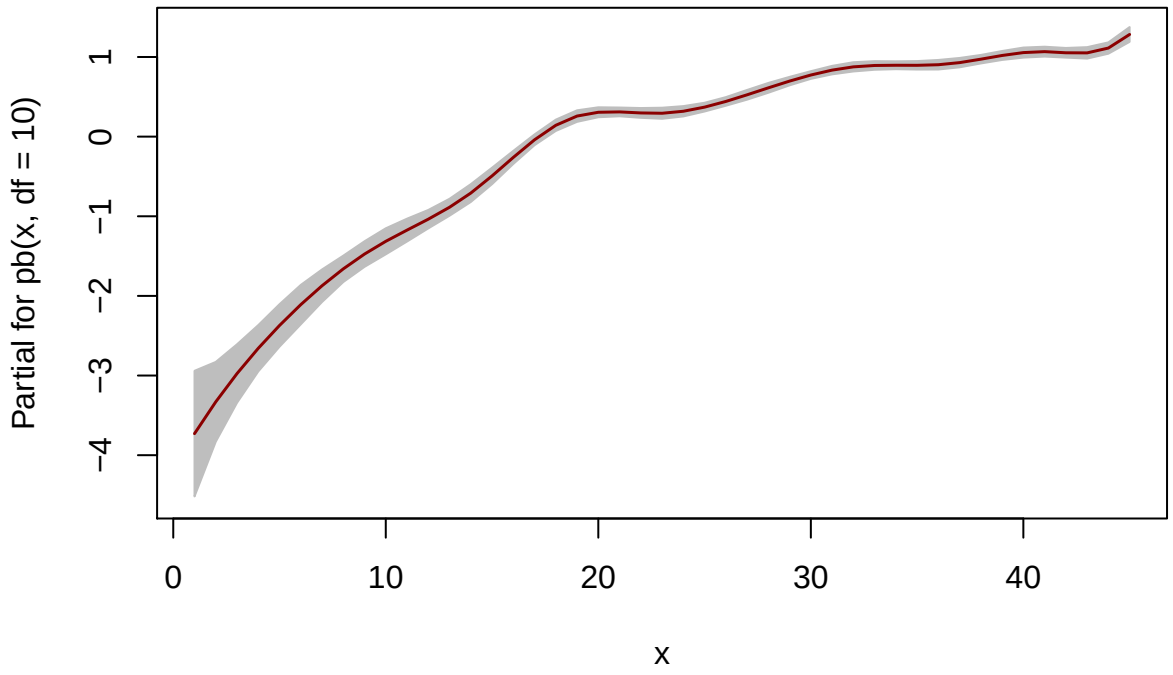
3 Plot regression terms for a specified parameter (term.plot)

Plots regression terms against their predictors, optionally with standard errors and partial residuals added. It is based on the R function termplot but is suitably changed to apply to GAMLSS objects.

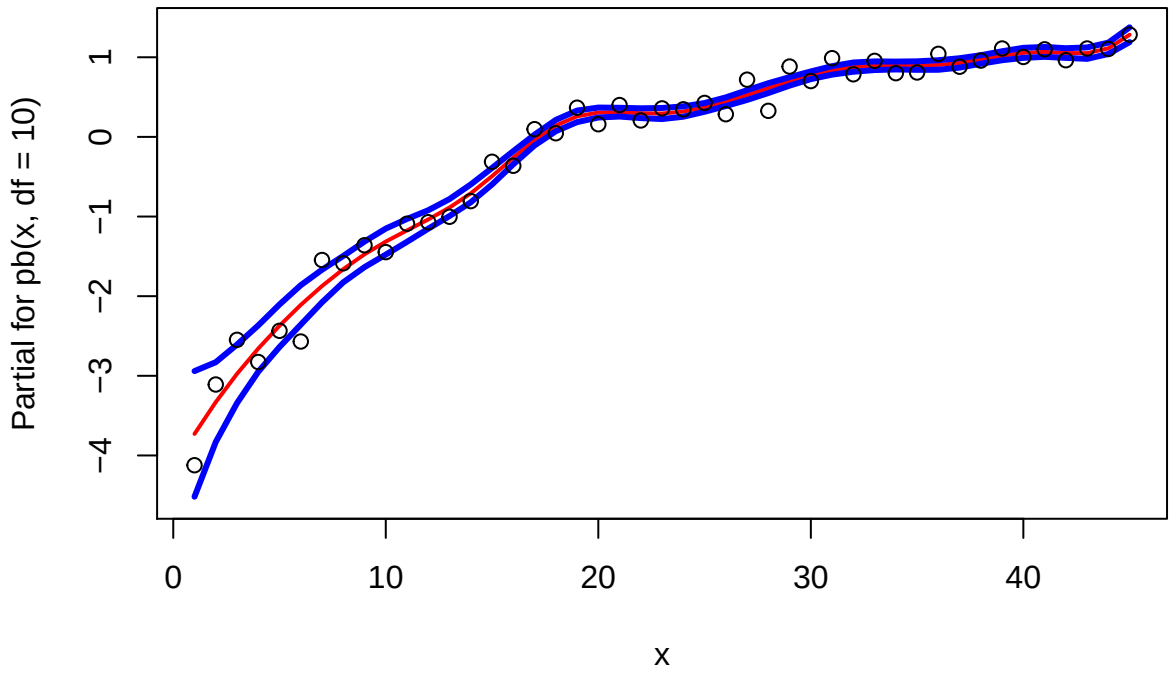
```
term.plot(object,
          what = c("mu", "sigma", "nu", "tau"),
          parameter= NULL,
          data = NULL,
          envir = environment(formula(object)),
          partial.resid = FALSE,
          rug = FALSE,
          terms = NULL,
          se = TRUE,
          ylim = c("common", "free"),
          scheme = c("shaded", "lines"),
          xlabs = NULL,
          ylabs = NULL,
          main = NULL,
          pages = 0,
          col.term = "darkred",
          col.se = "orange",
          col.shaded = "gray",
          col.res = "lightblue",
          col.rug = "gray",
          lwd.term = 1.5,
          lty.se = 2,
          lwd.se = 1,
          cex.res = 1,
          pch.res = par("pch"),
          ask = interactive() && nb.fig < n.tms && .Device != "postscript",
          use.factor.levels = TRUE,
          surface.gam = FALSE,
          polys = NULL,
          polys.scheme = "topo",...)
```

3.1 term.plot for aids dataset

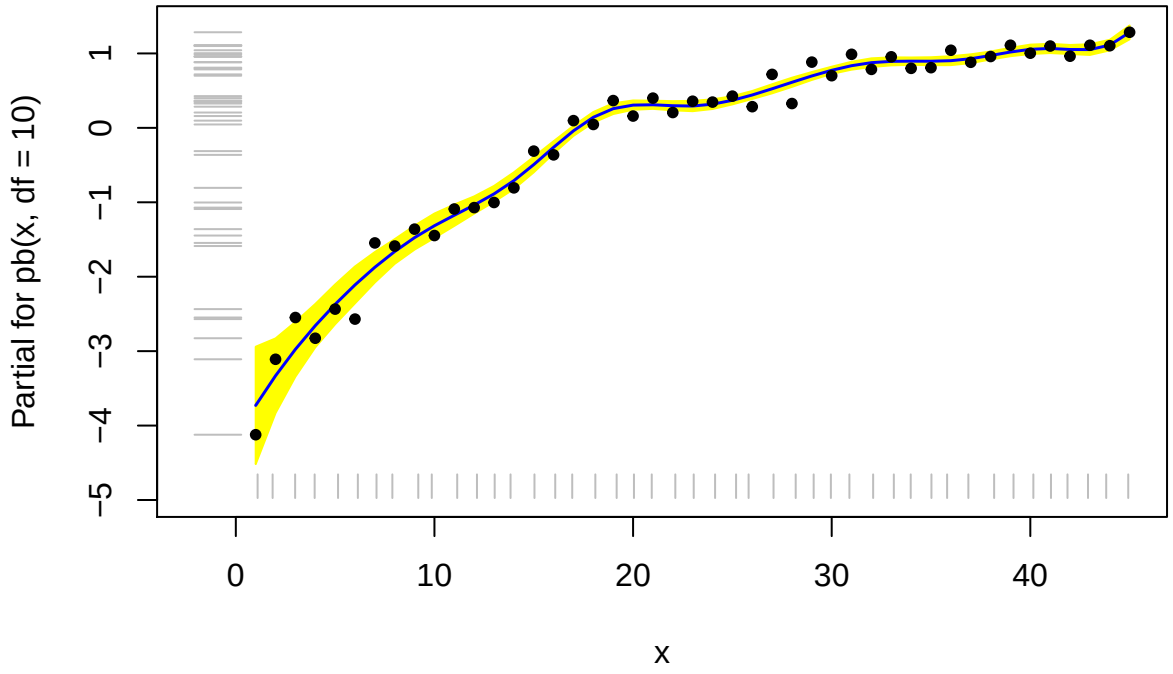
```
term.plot(aids1)
```




```
term.plot(aids1,
          partial.resid = TRUE,
          col.res="black",
          col.se="blue",
          lty.se=1,
          lwd.se = 3,
          lwd.term = 2,
          col.term="red",
          scheme="lines")
```



```
term.plot(aids1,
          partial.resid = TRUE,
          rug = TRUE,
          col.rug="gray",
          col.res="black",
          col.term="blue",
          col.shaded="yellow",
          pch=20,
          scheme="shaded")
```



4 Worm plot (wp)

Provides a single plot or multiple worm plots for a GAMLSS fitted or more general for any fitted models where the method resid() exist and the residuals are defined sensibly. The worm plot (a de-trended QQ-plot), van Buuren and Fredriks M. (2001), is a diagnostic tool for checking the residuals within different ranges (by default not overlapping) of the explanatory variable(s).

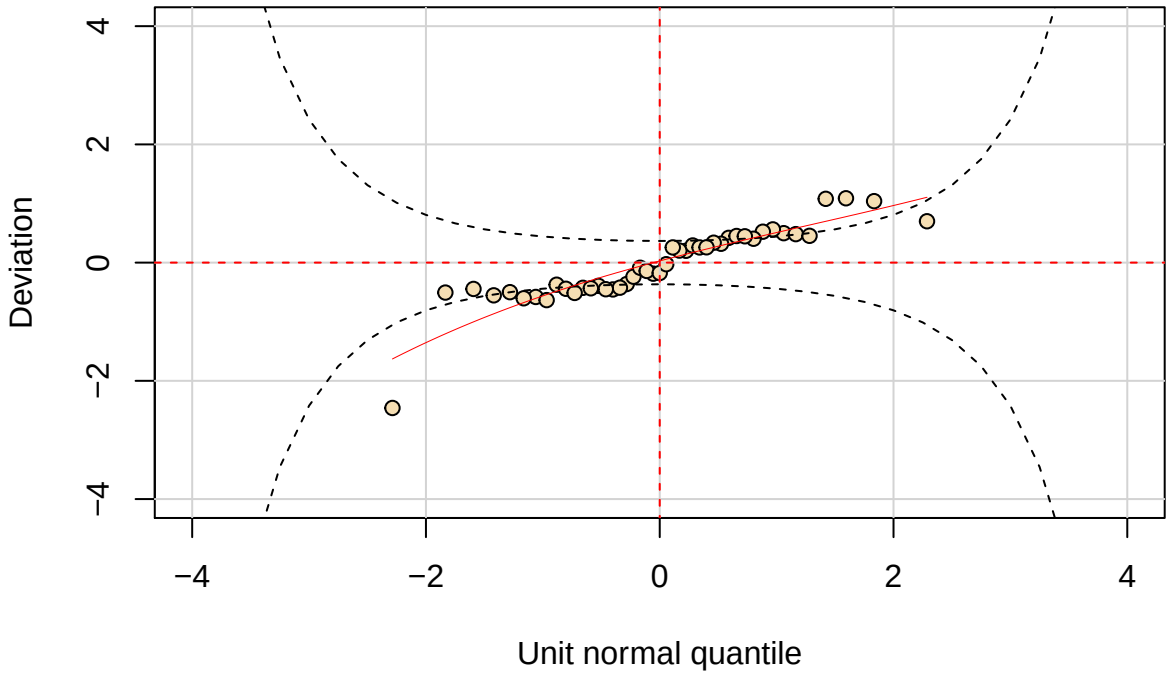
van Buuren and Fredriks M. (2001) Worm plot: simple diagnostic device for modelling growth reference curves. Statistics in Medicine, 20, 1259–1277


```
wp(object = NULL,
   xvar = NULL,
   resid = NULL,
   n.inter = 4,
   xcut.points = NULL,
   overlap = 0,
   xlim.all = 4,
   xlim.worm = 3.5,
   show.given = TRUE,
```

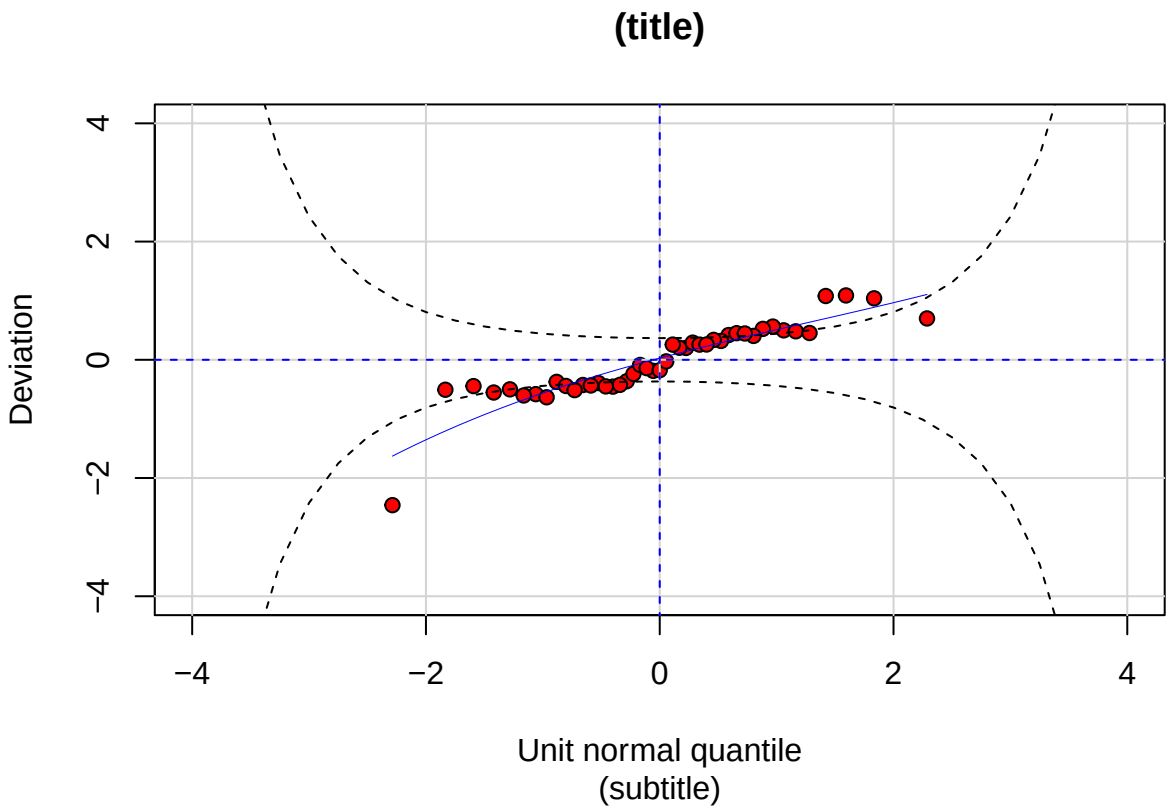
```
   line = TRUE,
   ylim.all = 12 * sqrt(1/length(resid)),
   ylim.worm = 12 * sqrt(n.inter/length(resid)),
   cex = 1,
   cex.lab = 1,
   pch = 21,
   bg = "wheat",
   col = "red",
   bar.bg = c(num = "light blue"), ...)
```

4.1 Worm plot for aids dataset

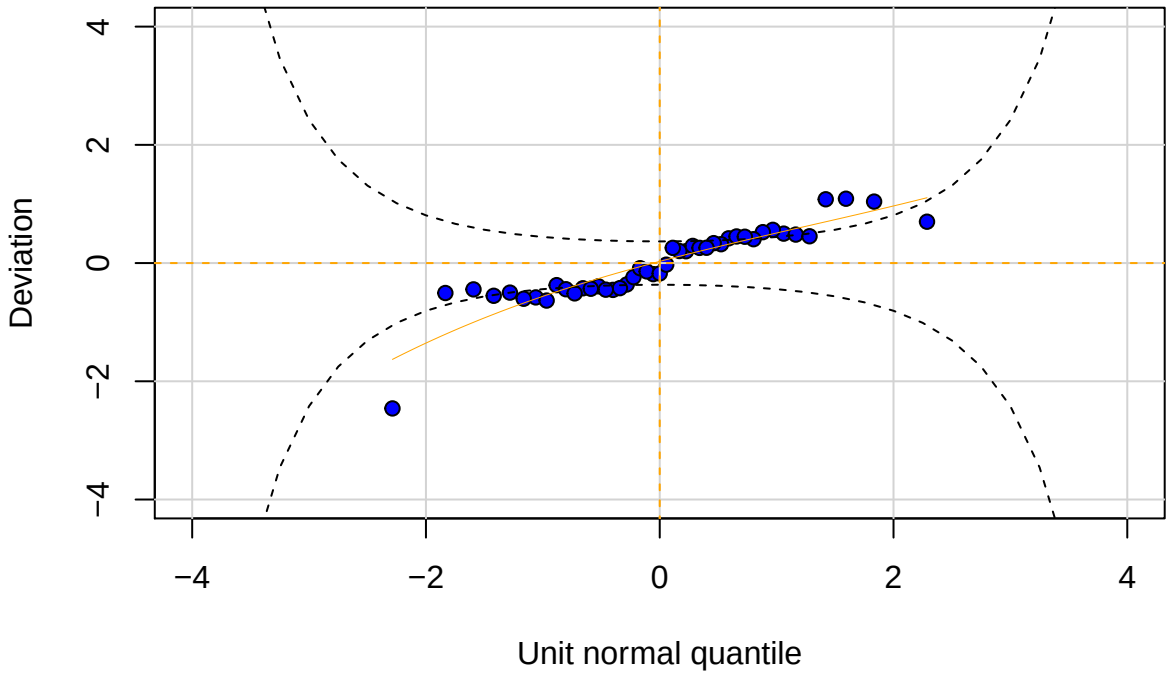
```
wp(aids1,ylim.all=4)
```



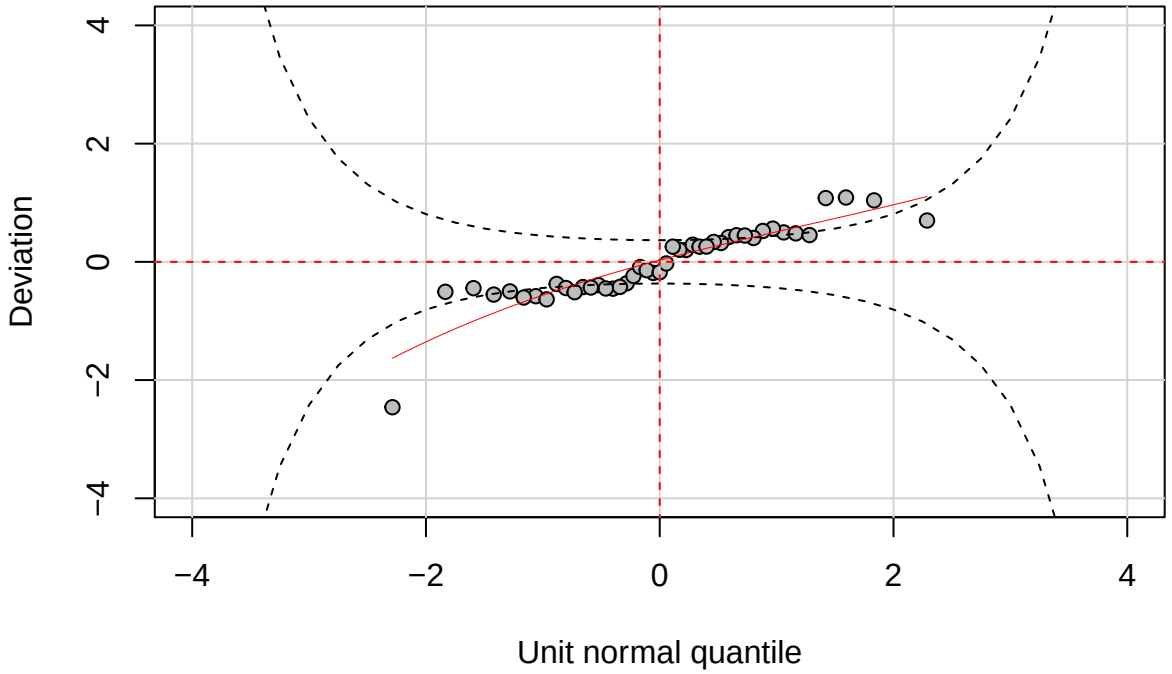

```
wp(aids1,
   ylim.all=4,
   col="blue",
   bg="red")
title(main = "(title)",
      sub = "(subtitle)")
```




```
wp(aids1,
   ylim.all=4,
   col="orange",
   bg="blue")
```

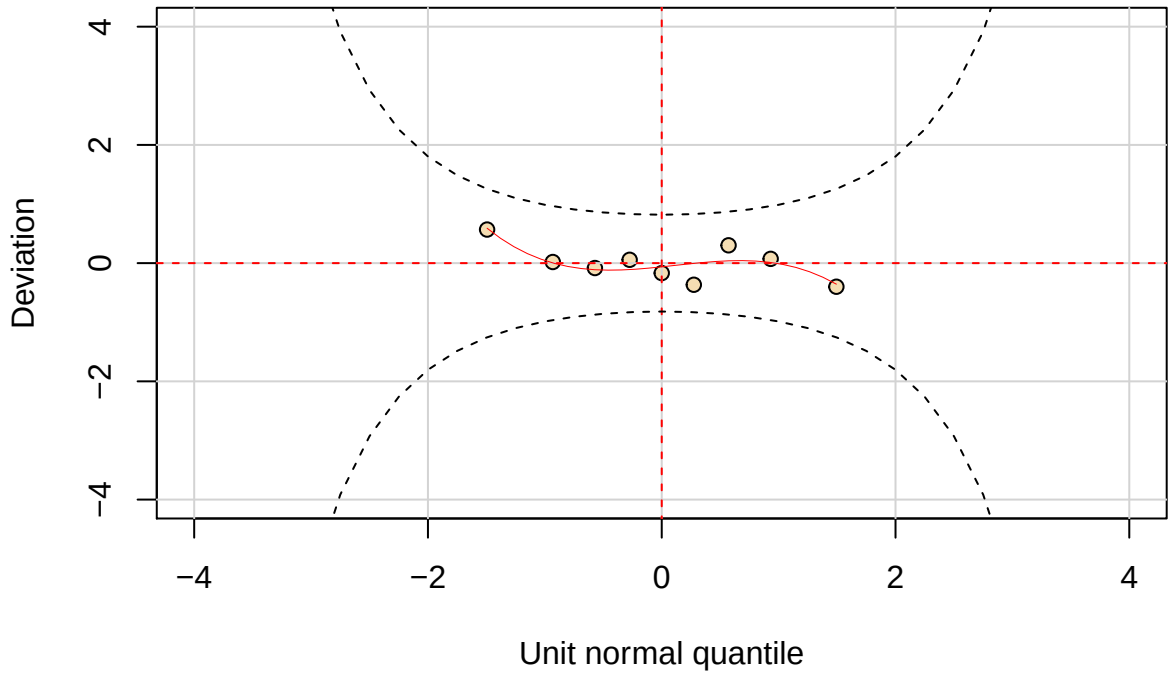


```
wp(aids1,
   ylim.all =4,
   col="red",
   bg="gray")
```



4.2 wp for glm object

```
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3,1,9)
treatment <- gl(3,3)

d.AD <- data.frame(treatment, outcome, counts)

glm.D93 <- glm(counts ~ outcome + treatment, family = poisson(link="log"))

wp(resid=resid(glm.D93, type="pearson"))
```




```
#gamlss.D93<- gamlss(counts ~ outcome + treatment, family = PO(mu.link="log"))
#wp(gamlss.D93)
```

5 Generalised Akaike Information Criterion (GAIC)

The function GAIC() calculates the Generalised Akaike information criterion (GAIC) for a given penalty k for a fitted GAMLSS object.

```
## GAIC functions
AIC(object, ...,
    k = 2,
    c = FALSE)

GAIC(object, ...,
     k = 2,
     c = FALSE)

GAIC.table(object, ...,
           k = c(2, 3.84,round(log(length(object$y)), 2)),
           text.to.show=NULL)

GAIC.scaled(object,..., k = 2, c = FALSE,
            plot = TRUE,
            text.cex = 0.7,
            which = 1,
            diff.dev = 1000,
            text.to.show = NULL,
            col = NULL,
            horiz = FALSE)
## S3 method for class 'gamlss'
```

```
extractAIC(fit,
           scale,
           k = 2,
           c = FALSE, ...)
```

5.1 GAIC for aids dataset

```
#GAIC.table(aids1,k=c(0,2,round(log(length(aids$y)),5)))
#aids1$G.deviance#the global deviance k=0
#aids1$aic#the Akaike information criterion k=2
#aids1$sbc#the Bayesian information criterion k=log(length(aids$y))
GAIC.table(aids1,aids2,aids3,k=c(0,2,round(log(length(aids$y)),2)))
```

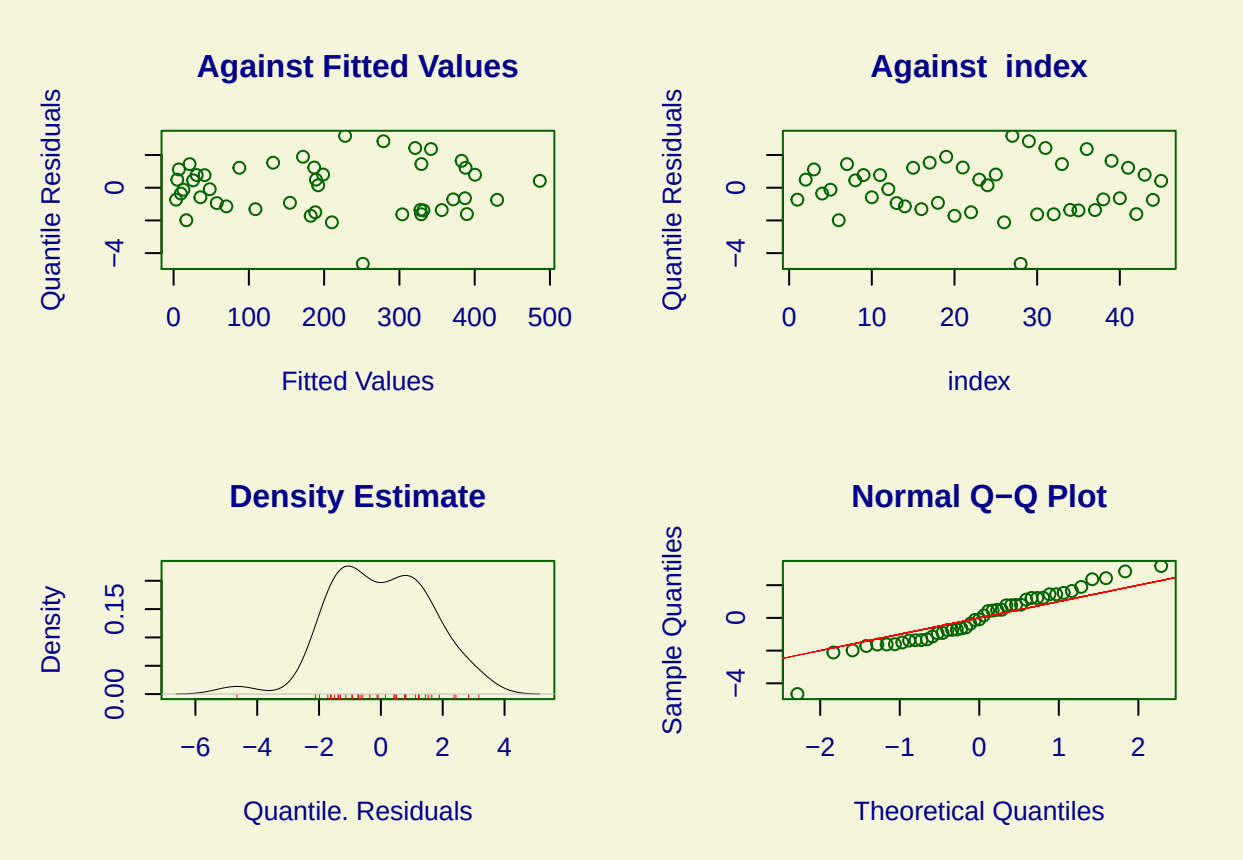
```
## minimum GAIC(k= 0 ) model: aids1
## minimum GAIC(k= 2 ) model: aids1
## minimum GAIC(k= 3.81 ) model: aids1

##              df     k=0      k=2   k=3.81
## aids1 12.000000 403.771 427.7710 449.4910
## aids2  7.000001 437.026 451.0260 463.6960
## aids3  5.000012 457.196 467.1961 476.2461
```

```
#GAIC(aids1, aids2, aids3, k=0)#Deviance of GAMLSS
#GAIC(aids1, aids2, aids3, k=2)#AIC
#GAIC(aids1, aids2, aids3,k=round(log(length(aids$y)),2))#BIC
```

6 Plot for gamlss object

6.1 Plot for aids data set

```
plot(aids1)
```




```
## ****************************************************************
##   Summary of the Randomised Quantile Residuals
##                           mean   =  -0.01159884
##                       variance   =  2.44423
##                coef. of skewness  =  -0.219995
##                coef. of kurtosis  =  3.093056
## Filliben correlation coefficient  =  0.9834088
## ****************************************************************
```

```
#plot(aids2)
#plot(aids3)
```

7 Fitting Different Parametric (fitDist)

The function fitDist() is using the function gamlssML() to fit all relevant parametric gamlss.family distributions, specified by the argument type), to a single data vector (with no explanatory variables). The final marginal distribution is the one selected by the generalised Akaike information criterion with penalty k. The default is k=2 i.e AIC.

7.1 fitDist for aids dataset

```
ajusteDis<- fitDist(y,data=aids,
        k=2,
        type="counts",
        trace=FALSE)
```

```
##   |                                                                      |
##   Lapack routine dgesv: system is exactly singular: U[4,4] = 0
##   |                                                                      |=================
```

```
summary(ajusteDis)
```

```
## ******************************************************************
## Family:  c("DPO", "Double Poisson")
##
## Call:  gamlssML(formula = y, family = DIST[i])
##
## Fitting method: "nlminb"
##
##
## Coefficient(s):
##            Estimate  Std. Error  t value   Pr(>|t|)
## eta.mu     5.259192    0.143765  36.5819 < 2.22e-16 ***
## eta.sigma  4.975518    0.286424  17.3712 < 2.22e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##  Degrees of Freedom for the fit: 2 Residual Deg. of Freedom   43
## Global Deviance:     561.877
##              AIC:     565.877
##              SBC:     569.491
```

8 chooseDist()

The function chooseDist() is using the function update.gamlss() to fit all relevant parametric (conditional) gamlss.family distributions to a given fitted gamlss model. The output of the function is a matrix with rows the different distributions (from the argument type) and columns the different GAIC's (). The default argument for k are 2, for AIC, 3.84, for Chi square, and log(n) for BIC. No final model is given by the function like for example in fitDist(). The function getOrder() can be used to rank the columns of the resulting table (matrix). The final model can be refitted using update(), see the examples.

8.1 chooseDist for aids dataset

```
#escolherD <- chooseDist(aids1, type="counts")
#save.image("escolherD.Rdata")
load("escolherD.Rdata")
#Beta Negative Binomial (BNB)
#AIC=377.5358
head(escolherD,10)
```

```
##                 2       3.84       3.81
## PO       427.7710   449.8510   449.4910
## GEOM     542.1561   564.2360   563.8760
## GEOMo  11686.2536 11708.3336 11707.9736
## LG     36801.8266 36823.9066 36823.5466
## YULE     904.6616   946.8234   946.1360
## ZIPF   15133.5688 15155.6488 15155.2888
## WARING         NA         NA         NA
## GPO     2947.2296  2971.1497  2970.7597
## DPO      405.9070   429.8270   429.4370
## BNB      377.5358   403.2958   402.8758
```

```
#which.min(escolherD[,1])#AIC
#which.min(escolherD[,2])#k=3.84
#which.min(escolherD[,3])#BIC
escolherD[which.min(escolherD[,1]),1:3]
```

```
##        2     3.84     3.81
## 377.5358 403.2958 402.8758
```

9 How to implement a new distribution

9.1 The Birnbaum-Saunders distribution

Let be $Y \sim BS(\mu, \sigma)$, with $Y > 0$, $\mu > 0$ and $\sigma > 0$, then

$$f(y) = \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{1}{2}a_y^2\right\} A_y$$

with $a_y = \frac{1}{\mu}\left(\sqrt{\frac{y}{\sigma}} - \sqrt{\frac{\sigma}{y}}\right)$ and $A_y = \frac{y^{-3/2}(y+\sigma)}{2\mu\sqrt{\sigma}}$

```
BS <- function(mu.link ="log", sigma.link="log") {

mstats <- checklink("mu.link","BS",substitute(mu.link),c("inverse", "log", "identity", "own"))
dstats <- checklink("sigma.link","BS",substitute(sigma.link),c("inverse", "log", "identity", "own"))

# fitting information
structure(
list( family        = c("BS","Birnbaum-Saunders"),
    parameters      = list(mu=TRUE,sigma=TRUE),
    nopar       = 2,
    type            = "Continuous",

    mu.link         = as.character(substitute(mu.link)),
    sigma.link      = as.character(substitute(sigma.link)),

    mu.linkfun      = mstats$linkfun,
    sigma.linkfun   = dstats$linkfun,

    mu.linkinv      = mstats$linkinv,
    sigma.linkinv   = dstats$linkinv,
```

```
    mu.dr       = mstats$mu.eta,
    sigma.dr        = dstats$mu.eta,

  dldm    = function() (1/mu^3)*(y/sigma + sigma/y -2) - 1/mu, #first derivative l w/r to mu
  d2ldm2  = function() -2/mu^2,                              #E(second derivative l w/r to mu)
##d2ldm2  = function() {d2ldm2<- eval.parent(expression(-dldp^2)) },#Aproximation

  dldd    = function() 1/(y+sigma) - 1/(2*sigma)-(1/(2*mu^2))*(1/y - y/sigma^2), ##first derivative l w/
d2ldd2  = function() {d2ldd2 <- eval.parent(expression(-dldp^2))},#Aproximation E(second derivative l w/

  d2ldmdd = function() -(2+mu^2)/(mu^3*sigma),  #E(second derivative l w/r to sigma)
##d2ldmdd = function() {d2ldmdd<- eval.parent(expression(-dldp^2))},#Aproximation E(second derivative l

G.dev.incr = function(y,mu,sigma,...) GD<- -2*dBS(y, mu, sigma, log = TRUE),
rqres        = expression(rqres(pfun="pBS", type="Continuous",y=y, mu=mu, sigma=sigma)),

    mu.initial      = expression({ mu <- (y+mean(y))/2 }),
    sigma.initial   = expression({ sigma <- rep(sd(y),length(y))}),

    mu.valid        = function(mu) all(mu > 0),
    sigma.valid     = function(sigma) all(sigma > 0),
    y.valid         = function(y) all(y > 0) ),

# the class definition
class = c("gamlss.family","family"))
}

#   the pdf function
dBS<- function(y, mu = 1, sigma = 1, log = FALSE)
{
    at <- (1/mu) * ( sqrt(y/sigma) - sqrt(sigma/y) )
    At <- y^(-3/2) * (y + sigma)/( 2 * mu * sqrt(sigma) )
    density  <- dnorm(at, 0, 1)*At

    if (log == TRUE) {
      density <- log(density)
    }
    return(density)
}

#   the cdf function
pBS<- function( q, mu = 1, sigma = 1, lower.tail = TRUE, log.p = FALSE)
{

    if (any(mu <= 0)) {
        stop("mu must be positive")
        }

    if (any(sigma <= 0)) {
        stop("sigma must be positive")
    }

    cumulative <- function(value, a, b) {
```

```
    integration<- integrate(dBS, lower = 0, upper = value, mu = a, sigma = b)$value
    return(integration)
    }

    cdf <- mapply(cumulative, q, a = mu, b = sigma)

    if (lower.tail == FALSE) {
        cdf <- (1 - cdf)
    }
    if (log.p == TRUE) {
        cdf <- log(cdf)
    }
    return(cdf)
}

#    the inverse cdf function
qBS<- function(p, mu = 1, sigma = 1, lower.tail = TRUE, log.p = FALSE)
{
   if (any(mu <= 0)) {
        stop("mu must be positive")
    }
    if (any(sigma <= 0)) {
        stop("sigma must be positive")
    }
    if (log.p == TRUE) {
        p <- log(p)
    }
    if (lower.tail == FALSE) {
        p <- (1 - p)
    }
    q <- sigma * (    (    (mu * qnorm(p, 0, 1)/2) + sqrt(   ((mu * qnorm(p, 0, 1)/2)^2) + 1)    )
    return(q)

}

#   the random number generation function
 rBS<- function(n, mu = 1, sigma = 1)
{
    if (any(mu <= 0))
        stop("mu must be positive")
    if (any(sigma <= 0))
        stop("sigma must be positive")

    z <- rnorm(n, 0, 1)
    t <- sigma*( 0.5*mu*z  +   sqrt( (0.5*mu*z)^2 +1 )  )^2     #sigma * (1 + (((mu^2) * (z^2))/2) + (mu
    return(t)
}
```

9.2 Fitting Binbaum-Saunders distribution

```
tempos <-  c(11, 11, 12, 13, 14, 14, 14, 14, 14, 15, 15, 15, 15, 16, 16, 16, 16, 16, 17,
       17, 17, 17, 18, 18, 18, 18, 19, 19, 19, 20, 20, 20, 20, 20, 20, 20, 20, 21,
```

```
       21, 21, 21, 22, 22, 22, 23, 23, 23, 23, 24, 24, 24, 24, 24, 25, 25, 25, 25,
       25, 25, 26, 26, 26, 26, 26, 27, 27, 27, 27, 27, 28, 28, 29, 29, 29, 29, 29,
       30, 30, 31, 31, 31, 31, 31, 31, 31, 32, 32, 32, 33, 33, 34, 34, 34, 34, 35,
       36, 36, 37, 37, 39, 39, 40, 40, 40, 41, 41, 42, 43, 43, 44, 44, 44, 46, 46,
       46, 47, 47, 48, 50, 50, 51, 52, 53, 54, 55, 56, 57, 59, 61, 66, 70, 89, 97)

dados<- data.frame(t=tempos)

ajuste_tempos<- gamlss(t~1,
                       family = BS(mu.link = "log",sigma.link = "log"),
                       data=dados, trace=FALSE)

fitted(ajuste_tempos,"mu")[1]
```

```
##        1
## 0.453605
```

```
fitted(ajuste_tempos,"sigma")[1]
```

```
##        1
## 27.87622
```

```
summary(ajuste_tempos)
```

```
## ******************************************************************
## Family:  c("BS", "Birnbaum-Saunders")
##
## Call:  gamlss(formula = t ~ 1, family = BS(mu.link = "log",
##     sigma.link = "log"), data = dados, trace = FALSE)
##
## Fitting method: RS()
##
## ------------------------------------------------------------------
## Mu link function:  log
## Mu Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.79053    0.06131  -12.89   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## ------------------------------------------------------------------
## Sigma link function:  log
## Sigma Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.32777    0.03832   86.85   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## ------------------------------------------------------------------
## No. of observations in the fit:  133
## Degrees of Freedom for the fit:  2
##       Residual Deg. of Freedom:  131
```

```
##                       at cycle:  3
##
## Global Deviance:     1044.945
##              AIC:     1048.945
##              SBC:     1054.726
## ******************************************************************
```

10 gamlss.family()

10.1 The Birnbaum Saunders distribution

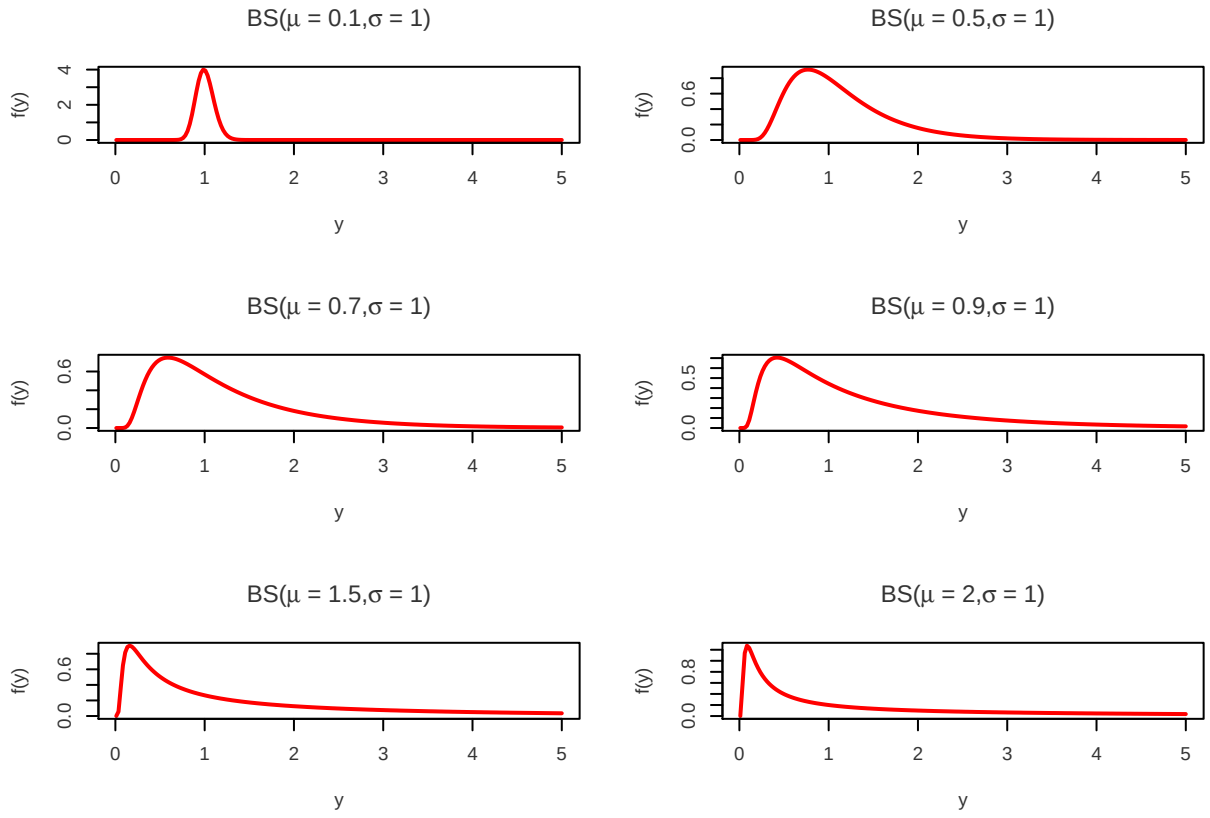
```
BS()
```

```
##
## GAMLSS Family: BS Birnbaum-Saunders
## Link function for mu    : log
## Link function for sigma: log
```

```
show.link(family = "BS")
```

```
## $mu
## c("inverse", "log", "identity", "own")
##
## $sigma
## c("inverse", "log", "identity", "own")
```

```
pdf.plot(family = BS(),
         mu=c(0.1,0.5,0.7,0.9,1.5,2), #alpha
         sigma=1, #beta
         min=0.01, max=5,
         col=2,lwd=2)
```



10.2 The SICHEL distribution

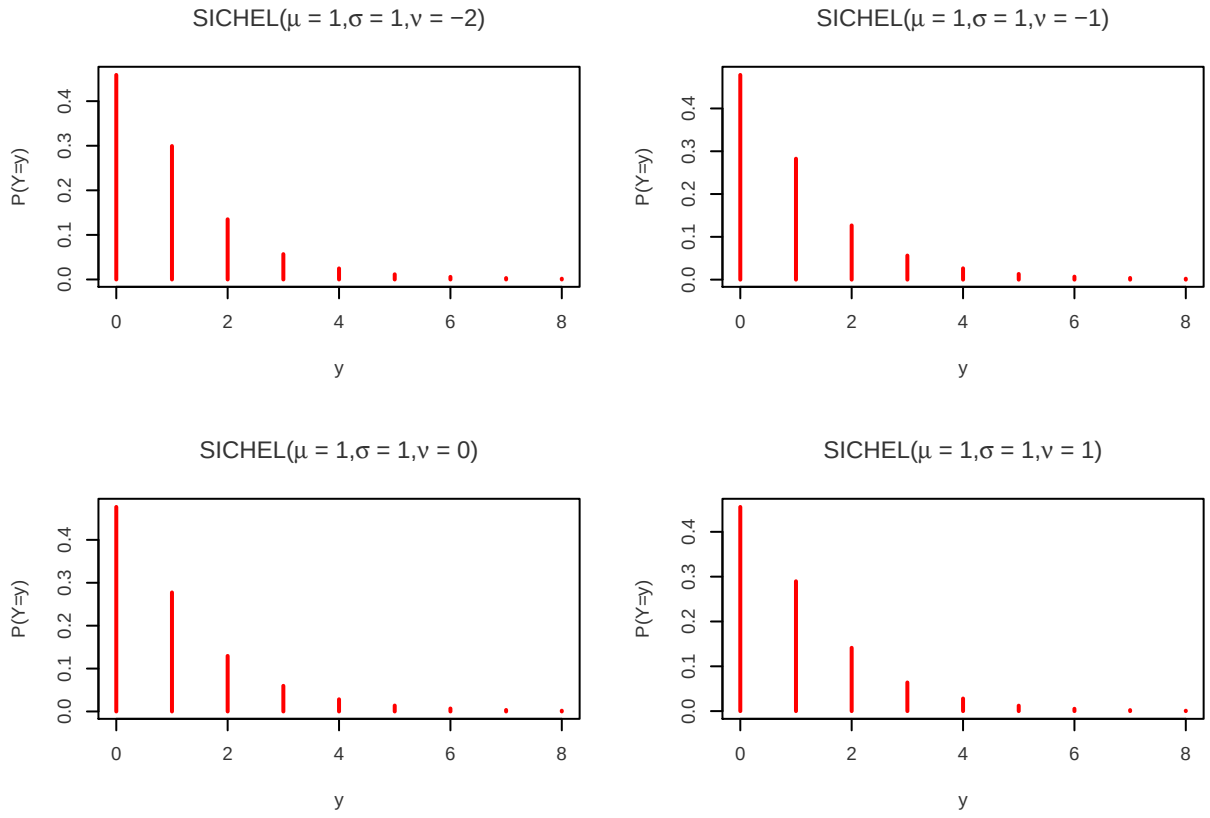
```
SICHEL()# gives information about the default links for the  Sichel distribution
```

```
##
## GAMLSS Family: SICHEL Sichel
## Link function for mu    : log
## Link function for sigma: log
## Link function for nu    : identity
```

```
show.link(family = "SICHEL")
```

```
## $mu
## c("1/mu^2", "log", "identity")
##
## $sigma
## c("inverse", "log", "identity")
##
## $nu
## c("1/nu^2", "log", "identity")
```

```
#plot the pdf using plot
pdf.plot(family = SICHEL(), mu=1, sigma=1,nu=-2:1, min=0, max=8, col=2,lwd=2)
```



11 The histogram and a fitted distribution to a variable (histDist)

11.1 histDist

This function fits constants to the parameters of a GAMLSS family distribution and them plot the histogram and the fitted distribution.

```
histDist(y, family = NO,
         freq = NULL,
         density = FALSE,
         nbins = 10,
         xlim = NULL,
         ylim = NULL,
         main = NULL,
         xlab = NULL,
         ylab = NULL,
         data = NULL,
         col.hist = "gray",
         border.hist = "blue",
         fg.hist = rainbow(12)[9],
         line.wd = 2,
         line.ty = c(1, 2),
         line.col = c(2, 3),
         col.main = "blue4",
         col.lab = "blue4",
         col.axis = "blue", ...)
```

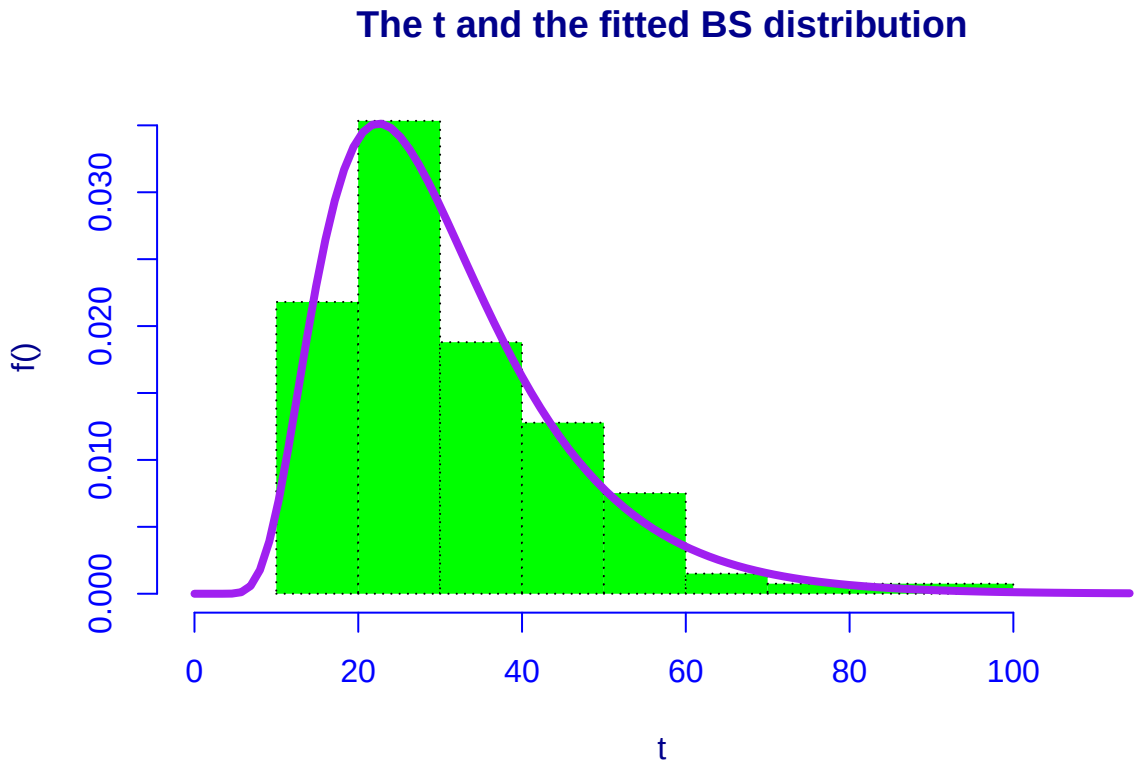
11.2 gamlssML

The function gamlssML() fits a gamlss.family distribution to single data set using a non linear maximisation algorithm in R. This is relevant only when explanatory variables do not exist.

```
gamlssML(formula, family = NO,
         weights = NULL, mu.start = NULL,
         sigma.start = NULL,
         nu.start = NULL,
         tau.start = NULL,
         mu.fix = FALSE,
         sigma.fix = FALSE,
         nu.fix = FALSE,
         tau.fix = FALSE,
         data, start.from = NULL, ...)
```

11.3 Binbaum-Saunders distribution

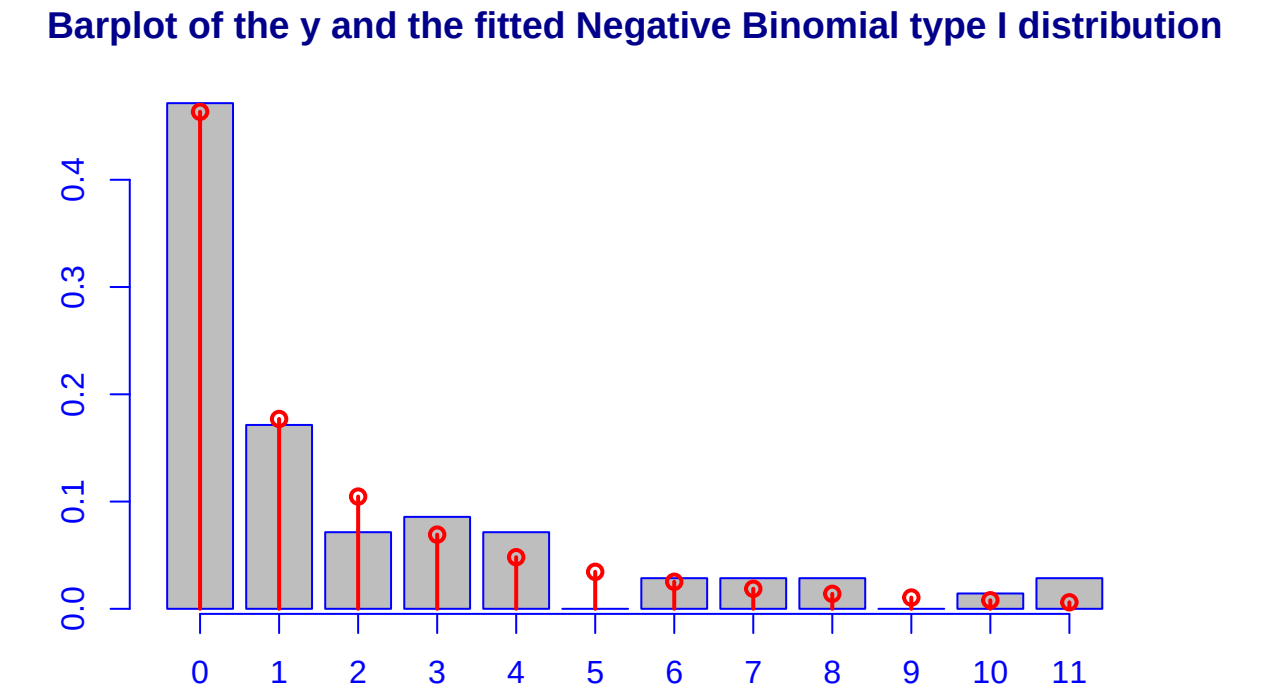
```
histDist(t, family = "BS",
         data=dados,
         col.hist="green",
         border.hist = "black",
         line.col = "purple",
         line.ty = 1,
         line.wd = 4,
         )
```



```
##
## Family:  c("BS", "Birnbaum-Saunders")
## Fitting method: "nlminb"
##
## Call:  gamlssML(formula = t, family = "BS", data = dados)
##
## Mu Coefficients:
## [1]  -0.7905
## Sigma Coefficients:
## [1]  3.328
##
##  Degrees of Freedom for the fit: 2 Residual Deg. of Freedom   131
## Global Deviance:     1044.95
##             AIC:     1048.95
##             SBC:     1054.73
```

11.4 Negative Binomial type I distribution

```
y <- c(0,1,2,3,4,6,7,8,10,11)
freq <- c(33,12,5,6,5,2,2,2,1,2)
histDist(y, "NBI", freq=freq)
```




```
##
## Family:  c("NBI", "Negative Binomial type I")
## Fitting method: "nlminb"
##
## Call:  gamlssML(formula = y, family = "NBI", weights = freq)
##
## Mu Coefficients:
```

```
## [1]  0.6493
## Sigma Coefficients:
## [1]  0.7397
##
##  Degrees of Freedom for the fit: 2 Residual Deg. of Freedom   8
## Global Deviance:     255.071
##             AIC:     259.071
##             SBC:     259.676
```

12 Choose a model by AIC in a Stepwise Algorithm (stepAIC, stepGAICAll.A, stepGAICAll.B, drop1All, add1All)

The function stepGAIC() performs stepwise model selection using a Generalized Akaike Information Criterion (GAIC). It is based on the function stepAIC() given in the library MASS of Venables and Ripley (2002). The function has been changed recently to allow parallel computation. The parallel computations are similar to the ones performed in the function boot() of the boot package. Note that since version 4.3-5 of gamlss the stepGAIC() do not have the option of using the function stepGAIC.CH through the argument additive.

Note that stepGAIC() is relying to the dropterm() and addterm() methods applyied to gamlss objects. drop1() and add1() are equivalent methods to the dropterm() and addterm() respectivelly but with different default arguments (see the examples).

The function stepGAIC.VR() is the old version of stepGAIC() with no parallel computations.

The function stepGAIC.CH is based on the S function step.gam() (see Chambers and Hastie (1991)) and it is more suited for model with smoothing additive terms when the degrees of freedom for smoothing are fixed in advance. This is something which rarely used these days, as most of the smoothing functions allow the calculations of the smoothing parameter, see for example the additive function pb()).

The functions stepGAIC.VR() and stepGAIC.CH() have been adapted to work with gamlss objects and the main difference is the scope argument, see below.

While the functions stepGAIC() is used to build models for individual parameters of the distribution of the response variable, the functions stepGAICAll.A() and stepGAICAll.A() are building models for all the parameters.

The functions stepGAICAll.A() and stepGAICAll.B() are based on the stepGAIC() function but use different strategies for selecting a appropriate final model.

stepGAICAll.A() has the following strategy:

Strategy A:

i) build a model for mu using a forward approach.

ii) given the model for mu build a model for sigma (forward)

iii) given the models for mu and sigma build a model for nu (forward)

iv) given the models for mu, sigma and nu build a model for tau (forward)

v) given the models for mu, sigma, nu and tau check whether the terms for nu are needed using backward elimination.

vi) given the models for mu, sigma, nu and tau check whether the terms for sigma are needed (backward).

vii) given the models for mu, sigma, nu and tau check whether the terms for mu are needed (backward).

Note for this strategy to work the scope argument should be set appropriately.

stepGAICAll.B() uses the same procedure as the function stepGAIC() but each term in the scope is fitted to all the parameters of the distribution, rather than the one specified by the argument what of stepGAIC(). The stepGAICAll.B() relies on the add1All() and drop1All() functions for the selection of variabes.

```
stepGAIC(object,
         scope,
         direction = c("both", "backward", "forward"),
         trace = T,
         keep = NULL,
         steps = 1000,
         scale = 0,
         what = c("mu", "sigma", "nu", "tau"),
         parameter= NULL, k = 2,
         parallel = c("no", "multicore", "snow"),
         ncpus = 1L,
         cl = NULL, ...)

stepGAICAll.A(object,
              scope = NULL,
              sigma.scope = NULL,
              nu.scope = NULL,
              tau.scope = NULL,
              mu.try = TRUE,
              sigma.try = TRUE,
              nu.try = TRUE,
              tau.try = TRUE,
              parallel = c("no", "multicore", "snow"),
              ncpus = 1L,
              cl = NULL,  ...)


stepGAICAll.B(object,
              scope,
              direction = c("both", "backward", "forward"),
              trace = T,
              keep = NULL,
              steps = 1000,
              scale = 0,
              k = 2,
              parallel = c("no", "multicore", "snow"),
              ncpus = 1L,
              cl = NULL, ...)

drop1All(object,
         scope,
         test = c("Chisq", "none"),
         k = 2,
         sorted = FALSE,
         trace = FALSE,
         parallel = c("no", "multicore", "snow"),
         ncpus = 1L, cl = NULL, ...)
```

```
add1All(object,
        scope,
        test = c("Chisq", "none"),
        k = 2,
        sorted = FALSE,
        trace = FALSE,
        parallel = c("no", "multicore", "snow"), ncpus = 1L, cl = NULL, ...)
```

12.1 Risk Factors Associated with Low Infant Birth Weight (birthwt)

```
example(birthwt)
```

```
birthwt.gamlss <- gamlss(low ~ .,
                         family = BI(mu.link="logit"),
                         data = bwt)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 195.4755
## GAMLSS-RS iteration 2: Global Deviance = 195.4755
```

```
birthwt.step <- stepGAIC(birthwt.gamlss,
                         trace = TRUE,
                         k=2)
```

```
## Distribution parameter:  mu
## Start:  AIC= 217.48
##  low ~ age + lwt + race + smoke + ptd + ht + ui + ftv
##
##         Df    AIC
## - ftv    2 214.83
## - age    1 216.42
## <none>     217.48
## - ui     1 217.59
## - smoke  1 218.67
## - race   2 219.23
## - lwt    1 220.95
## - ht     1 222.93
## - ptd    1 223.58
##
## Step:  AIC= 214.83
##  low ~ age + lwt + race + smoke + ptd + ht + ui
##
##         Df    AIC
## - age    1 213.85
## <none>     214.83
## - ui     1 215.15
## - race   2 217.24
## - smoke  1 217.25
## - lwt    1 217.83
## - ptd    1 219.95
## - ht     1 220.01
```

```
##
## Step:  AIC= 213.85
##  low ~ lwt + race + smoke + ptd + ht + ui
##
##         Df    AIC
## <none>     213.85
## - ui     1 214.48
## - smoke  1 216.57
## - race   2 217.47
## - lwt    1 217.82
## - ptd    1 218.22
## - ht     1 219.16
```

13 Plotting a simple GAMLSS model for demonstration purpose

This is to plot a simple GAMLSS model where only one explanatory variable exist in order to demonstrated how the distribution of the response changes according to values of the explanatory variable.

```
plotSimpleGamlss(y, x,
                 model = NULL,
                 formula = NULL,
                 data = NULL,
                 family = NULL,
                 val = NULL,
                 N = 1000,
                 x.val = quantile(x),
                 ylim = c(min(y), max(y)),
                 xlim = c(min(x), max(x)),
                 ...)
```

```
library(gamlss.util)
library(colorspace)

LO()
```

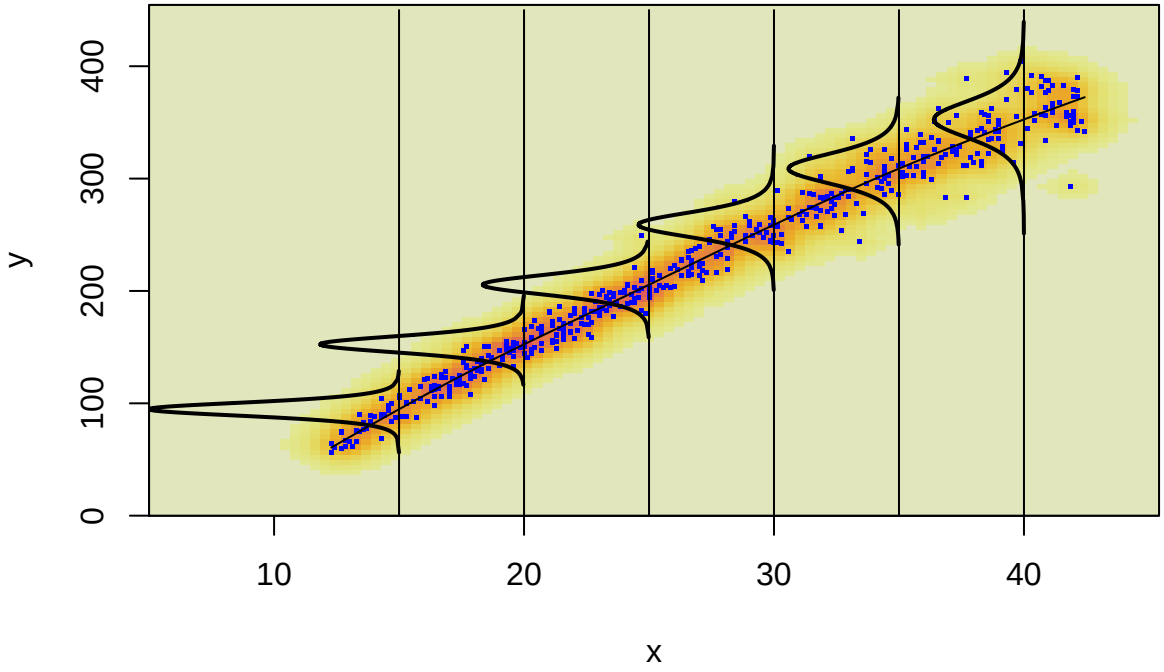
```
##
## GAMLSS Family: LO Logistic
## Link function for mu    : identity
## Link function for sigma: log
```

```
show.link(family="LO")
```

```
## $mu
## c("inverse", "log", "identity", "own")
##
## $sigma
## c("inverse", "log", "identity", "own")
```

```
m1 <- gamlss(y~pb(x),
             sigma.fo=~pb(x),
             data=abdom,
```

```
              family=LO,
              trace=FALSE)

plotSimpleGamlss(y,x,
                 model=m1,
                 data=abdom,
                 x.val=seq(15, 40, 5),
                 ylim=c(0, 450),
                 xlim=c(5, 45),
                 cols=heat_hcl(100))
```




```
## new prediction
## New way of prediction in pb()  (starting from GAMLSS version 5.0-3)
## new prediction
## New way of prediction in pb()  (starting from GAMLSS version 5.0-3)
```

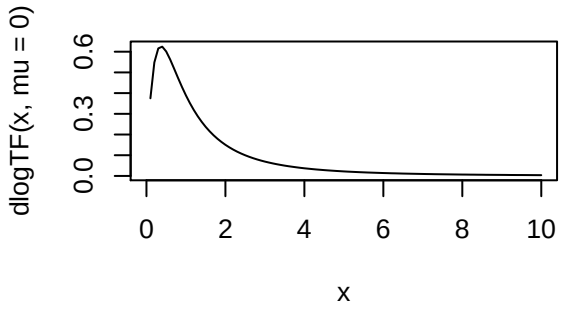
14 gen.Family

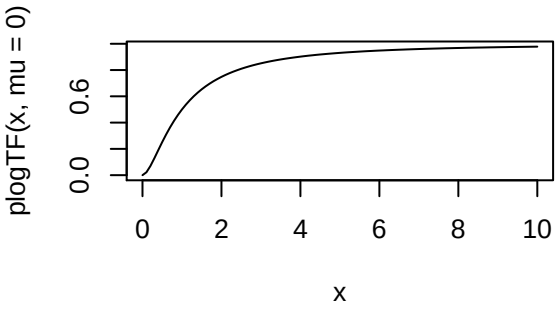
Functions to generate log and logit distributions from existing continuous gamlss.family distributions

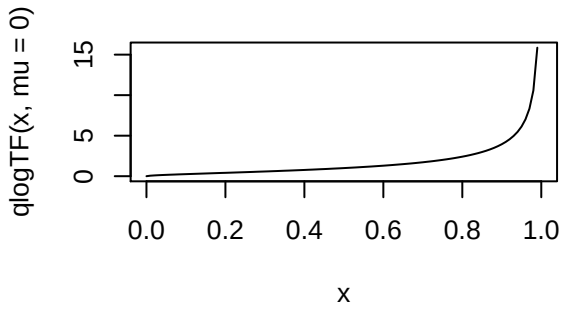
```
example(gen.Family)
```

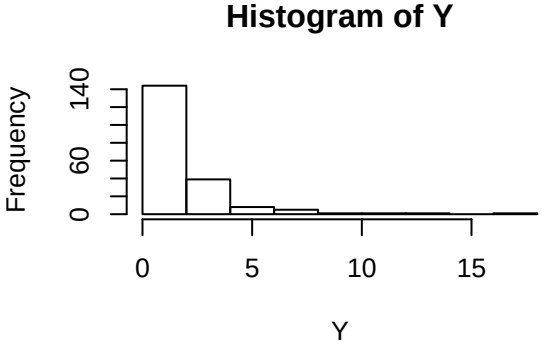
```
##
## gn.Fml> # generating a log t distribution
## gn.Fml> gen.Family("TF")
## A  log  family of distributions from TF has been generated
##  and saved under the names:
##  dlogTF plogTF qlogTF rlogTF logTF
##
```

```
## gn.Fml> # plotting the d, p, q, and r functions
## gn.Fml> op<-par(mfrow=c(2,2))
##
## gn.Fml> curve(dlogTF(x, mu=0), 0, 10)

##
## gn.Fml> curve(plogTF(x, mu=0), 0, 10)

##
## gn.Fml> curve(qlogTF(x, mu=0), 0, 1)

##
## gn.Fml> Y<- rlogTF(200)
##
## gn.Fml> hist(Y)
```

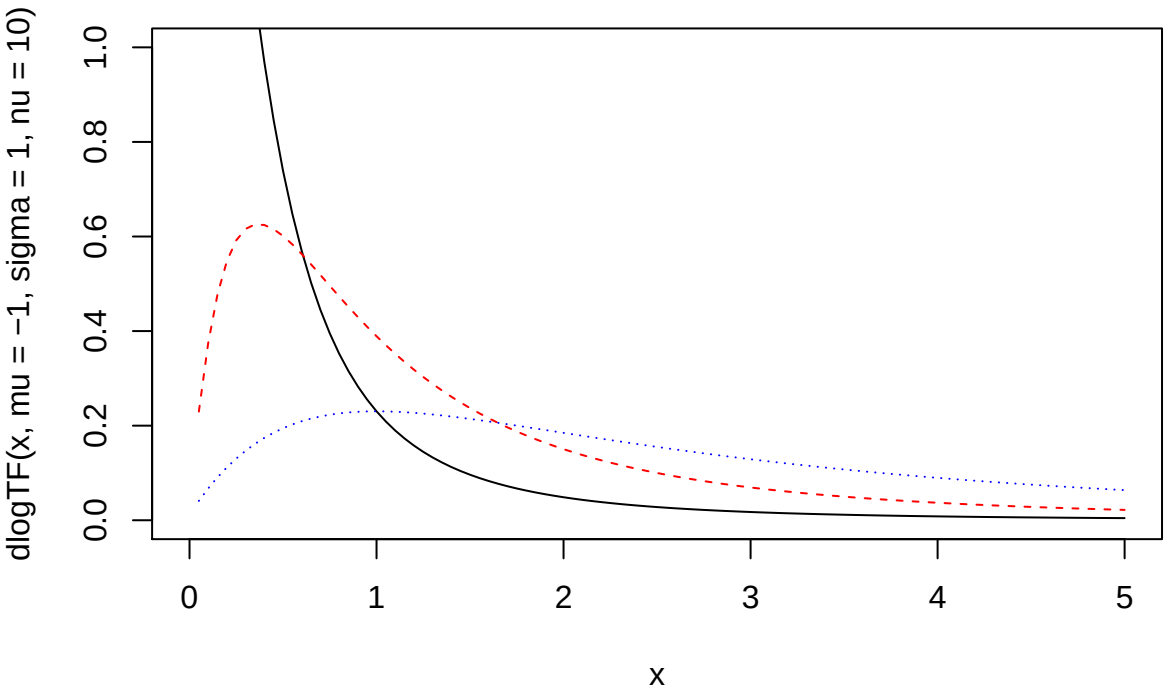




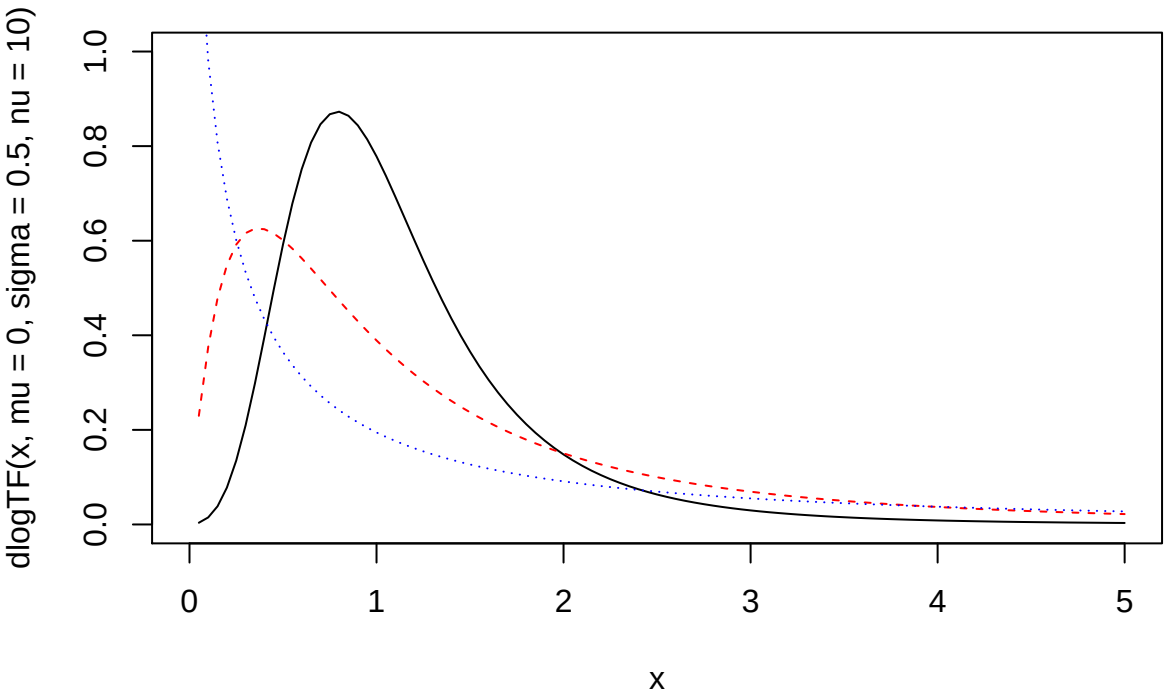





```
##
## gn.Fml> par(op)
##
## gn.Fml> # different mu
## gn.Fml> curve(dlogTF(x, mu=-1, sigma=1, nu=10), 0, 5, ylim=c(0,1))
```

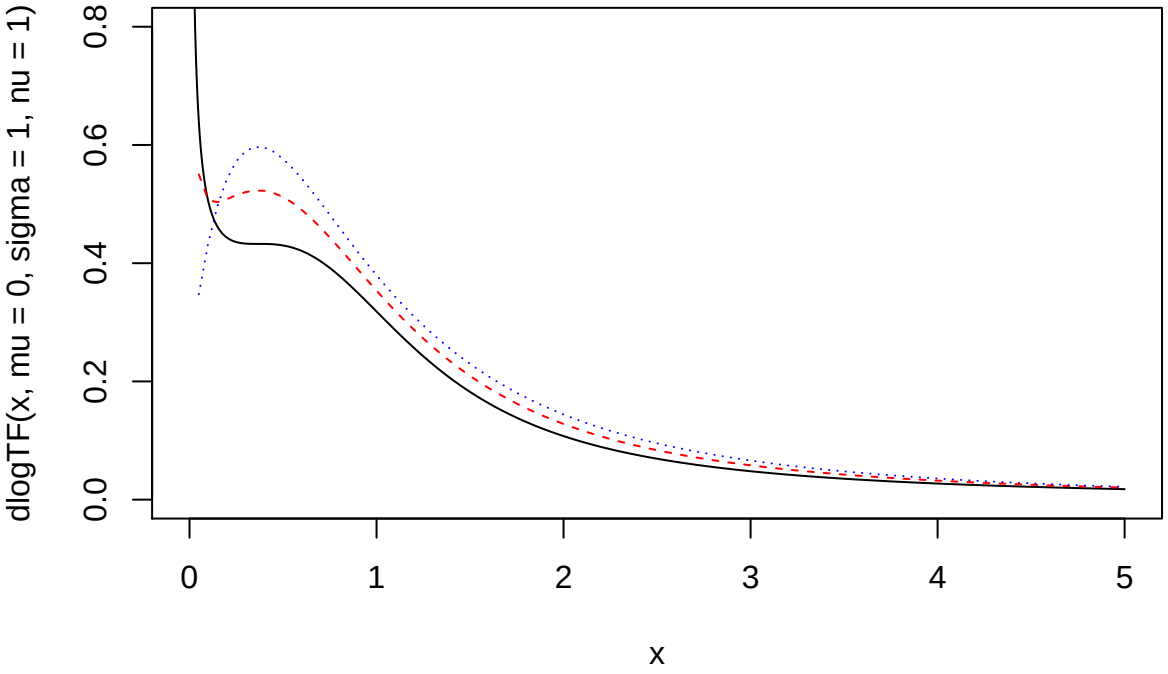



```
##
## gn.Fml> curve(dlogTF(x, mu=0, sigma=1, nu=10), 0,  5, add=TRUE, col="red", lty=2)
##
## gn.Fml> curve(dlogTF(x, mu=1, sigma=1, nu=10), 0, 5, add=TRUE, col="blue", lty=3)
##
## gn.Fml> # different sigma
## gn.Fml> curve(dlogTF(x, mu=0, sigma=.5, nu=10), 0, 5, ylim=c(0,1))
```



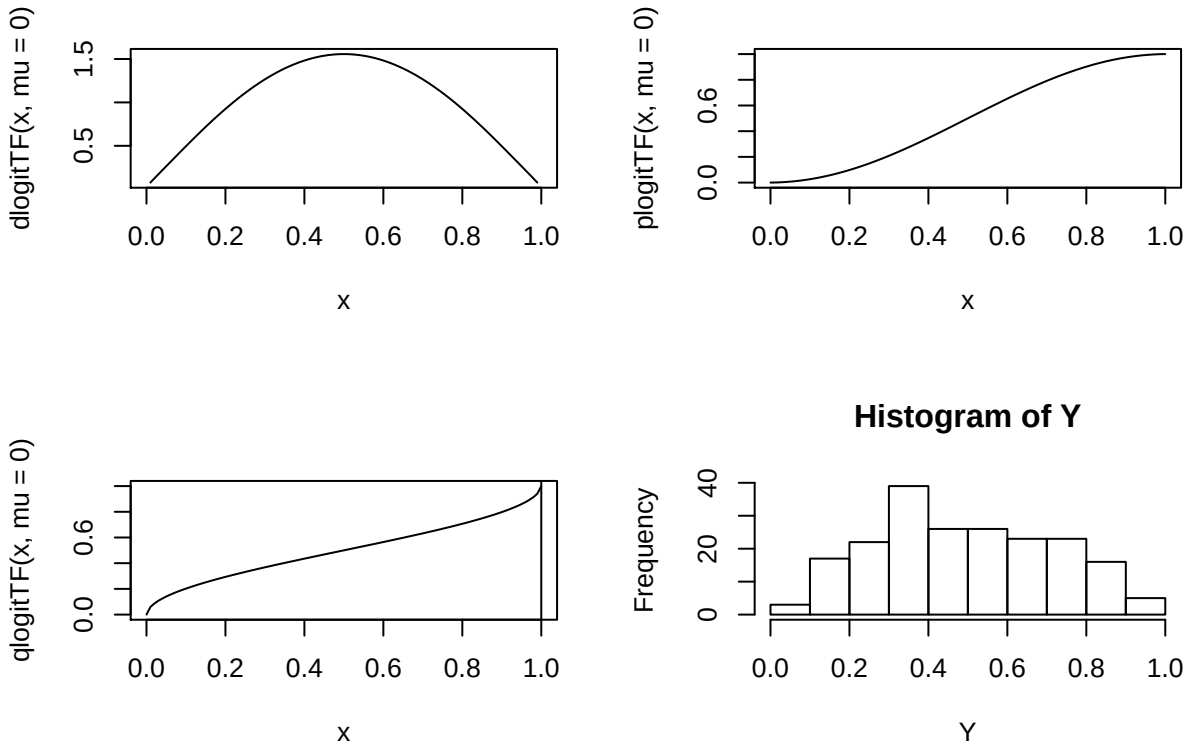

```
##
```

```
## gn.Fml> curve(dlogTF(x, mu=0, sigma=1, nu=10), 0, 5, add=TRUE, col="red", lty=2)
##
## gn.Fml> curve(dlogTF(x, mu=0, sigma=2, nu=10), 0, 5, add=TRUE, col="blue", lty=3)
##
## gn.Fml> # different degrees of freedom nu
## gn.Fml> curve(dlogTF(x,  mu=0, sigma=1, nu=1), 0, 5, ylim=c(0,.8), n = 1001)
```

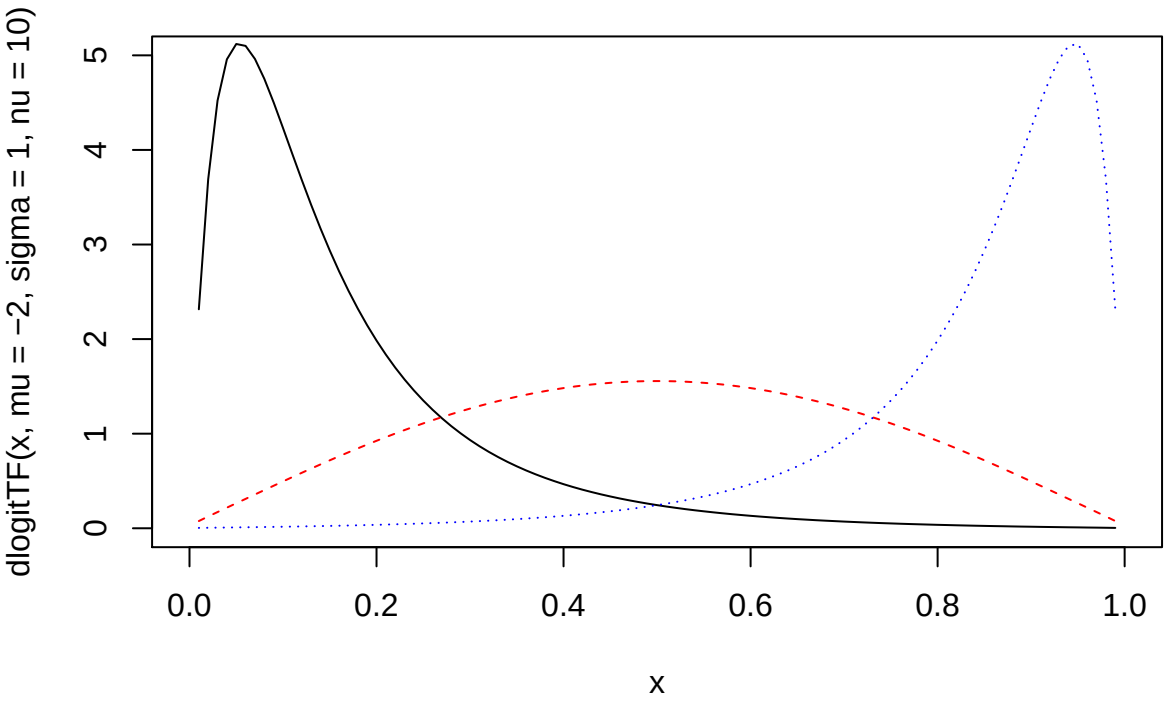



```
##
## gn.Fml> curve(dlogTF(x,  mu=0, sigma=1, nu=2), 0, 5, add=TRUE, col="red", lty=2)
##
## gn.Fml> curve(dlogTF(x,  mu=0, sigma=1, nu=5), 0, 5, add=TRUE, col="blue", lty=3)
##
## gn.Fml> # generating a logit t distribution
## gn.Fml> gen.Family("TF", "logit")
## A  logit  family of distributions from TF has been generated
##  and saved under the names:
##  dlogitTF plogitTF qlogitTF rlogitTF logitTF
##
## gn.Fml> # plotting the d, p, q, and r functions
## gn.Fml> op<-par(mfrow=c(2,2))
##
## gn.Fml> curve(dlogitTF(x, mu=0), 0, 1)

##
## gn.Fml> curve(plogitTF(x, mu=0), 0, 1)

##
## gn.Fml> curve(qlogitTF(x, mu=0), 0, 1)

##
## gn.Fml> abline(v=1)
```

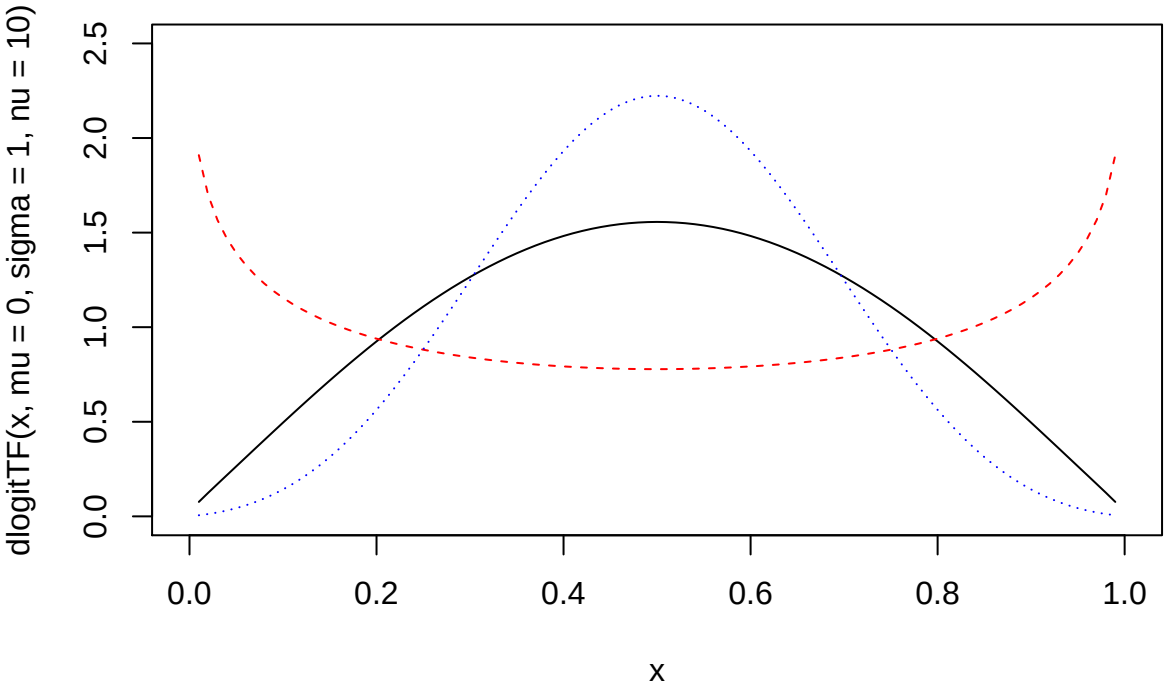
```
##
## gn.Fml> Y<- rlogitTF(200)
##
## gn.Fml> hist(Y)
```




```
##
## gn.Fml> par(op)
##
## gn.Fml> # different mu
## gn.Fml> curve(dlogitTF(x, mu=-2, sigma=1, nu=10), 0, 1, ylim=c(0,5))
```

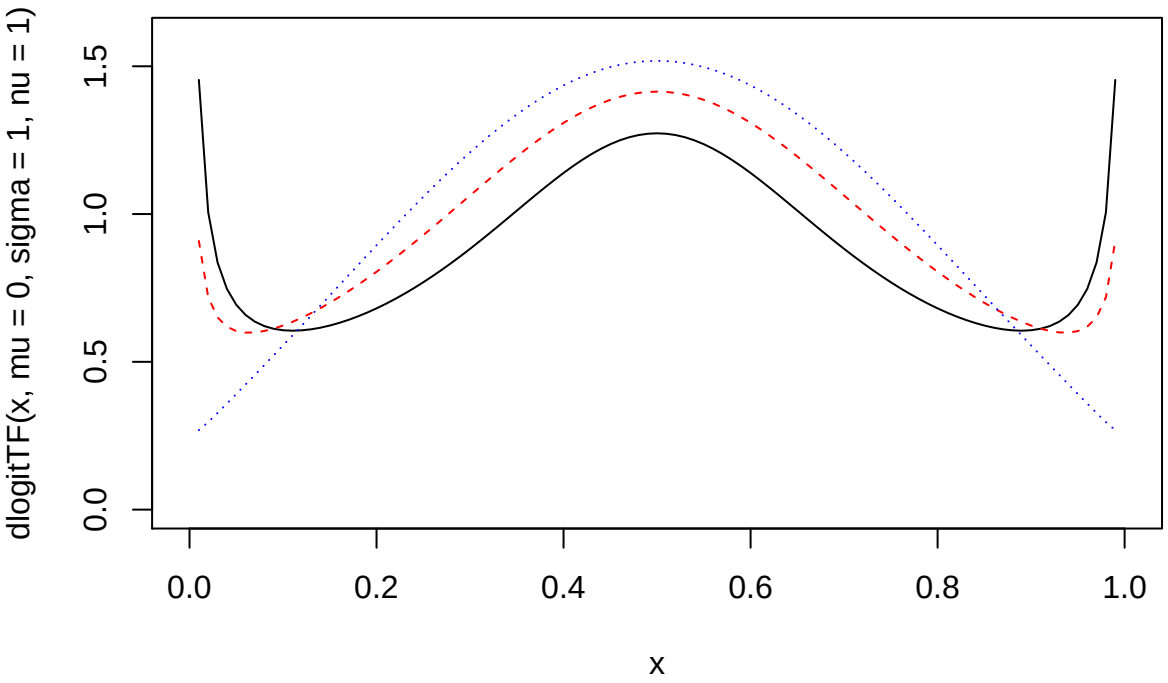



```
##
## gn.Fml> curve(dlogitTF(x, mu=0, sigma=1, nu=10), 0, 1, add=TRUE, col="red", lty=2)
##
## gn.Fml> curve(dlogitTF(x, mu=2, sigma=1, nu=10), 0, 1, add=TRUE, col="blue", lty=3)
##
## gn.Fml> # different sigma
## gn.Fml> curve(dlogitTF(x, mu=0, sigma=1, nu=10), 0, 1, ylim=c(0,2.5))
```




```
##
```

```
## gn.Fml> curve(dlogitTF(x, mu=0, sigma=2, nu=10), 0, 1, add=TRUE, col="red", lty=2)
##
## gn.Fml> curve(dlogitTF(x, mu=0, sigma=.7, nu=10), 0, 1, add=TRUE, col="blue", lty=3)
##
## gn.Fml> # different degrees of freedom nu
## gn.Fml> curve(dlogitTF(x,  mu=0, sigma=1, nu=1), 0, 1, ylim=c(0,1.6))
```




```
##
## gn.Fml> curve(dlogitTF(x,  mu=0, sigma=1, nu=2), 0, 1, add=TRUE, col="red", lty=2)
##
## gn.Fml> curve(dlogitTF(x,  mu=0, sigma=1, nu=5), 0, 1, add=TRUE, col="blue", lty=3)
```

15 momentSK

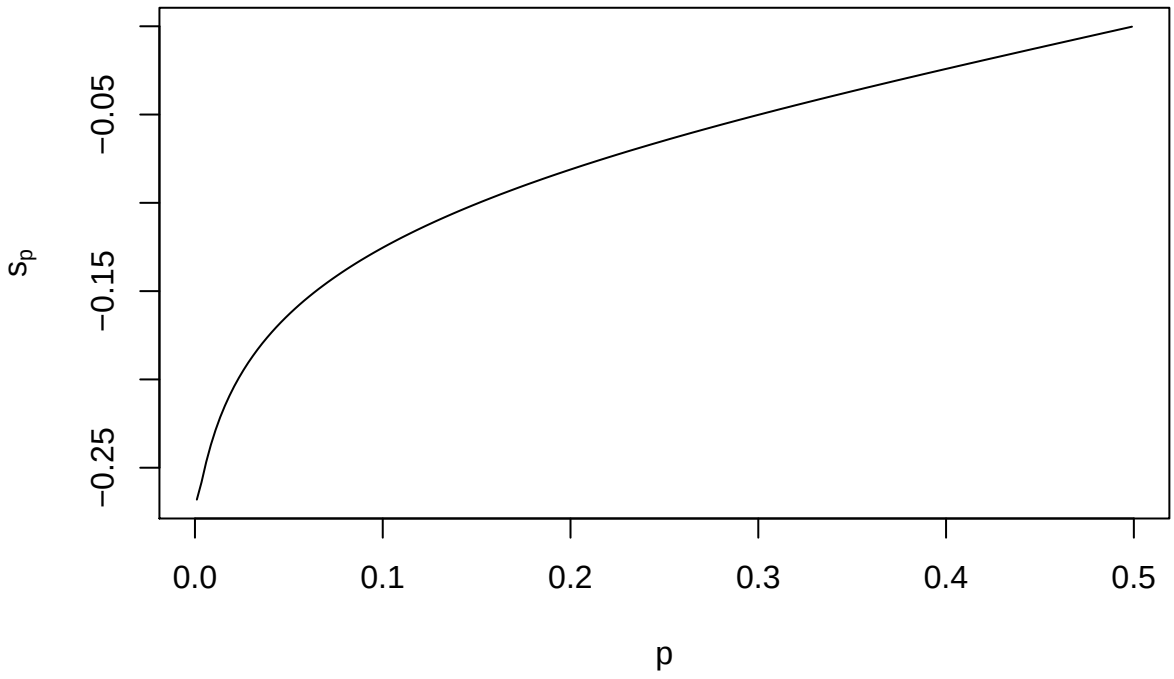
Sample and theoretical Moment and Centile Skewness and Kurtosis Functions

```
example(momentSK)
```

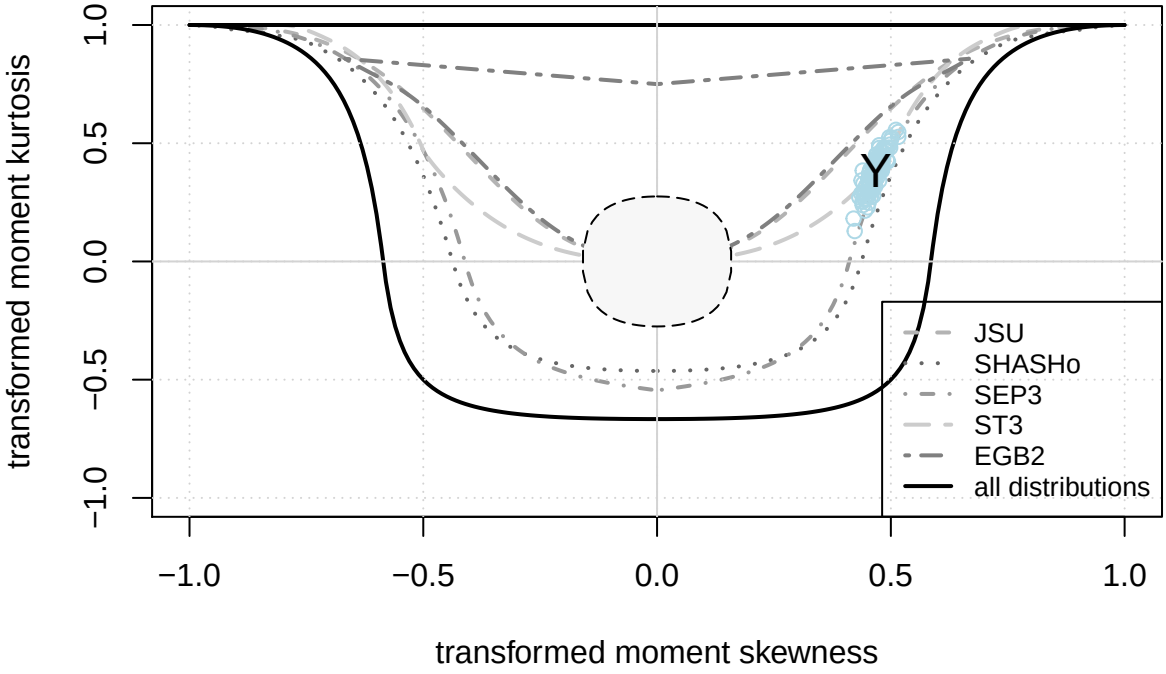
```
##
## mmntSK> Y <- rSEP3(1000)
##
## mmntSK> momentSK(Y)
## $mom.skew
## [1] 0.8798175
##
## $trans.mom.skew
## [1] 0.4680335
##
## $mom.kurt
```

```
## [1] 3.634732
##
## $excess.mom.kurt
## [1] 0.634732
##
## $trans.mom.kurt
## [1] 0.3882789
##
## $jarque.bera.test
## [1] 145.8
##
##
## mmntSK> centileSK(Y)
## $S0.25
##
## 0.1736059
##
## $S0.01
##
## 0.3938586
##
## $trans.S0.25
##
## 0.1479252
##
## $trans.S0.01
##
## 0.2825671
##
## $K0.01
##
## 3.320921
##
## $standK0.01
##
## 0.9628648
##
## $exc.K0.01
##
## -0.1280793
##
## $trans.K0.01
##
## -0.1135375
##
##
## mmntSK> centileSkew(Y, cent=20)
## $p
## [1] 0.2
##
## $Sp
##
## 0.1686247
##
```

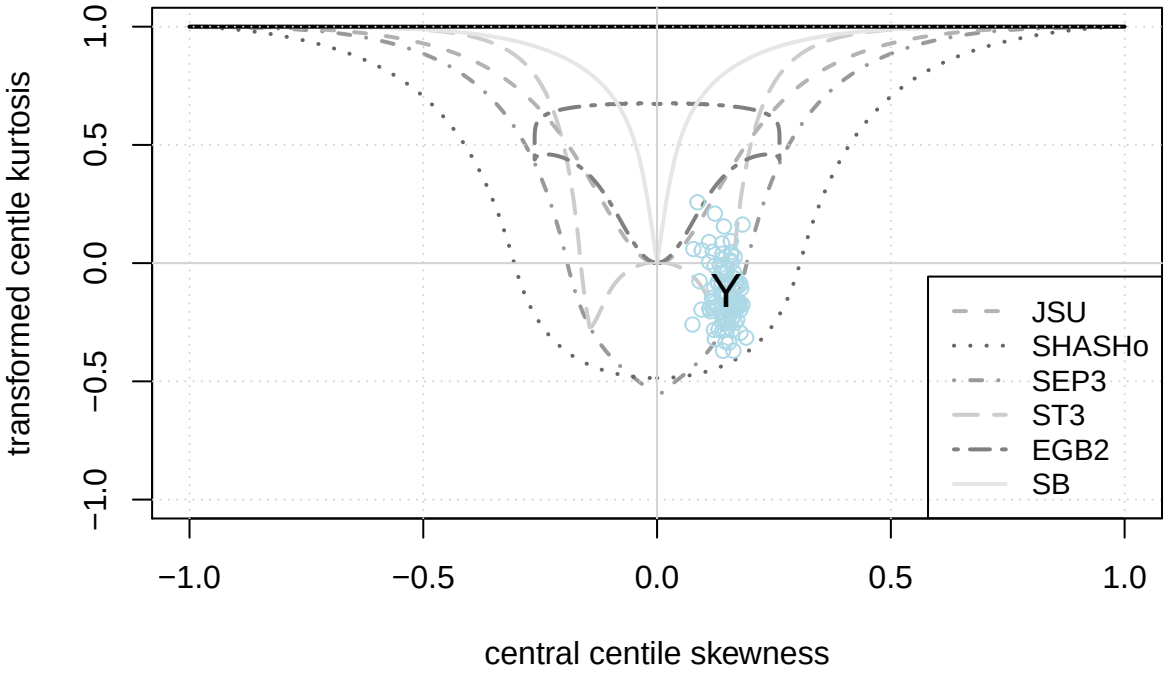
```
## $tSp
##
## 0.1442933
##
##
## mmntSK> centileKurt(Y, cent=30)
## $p
## [1] 0.3
##
## $Kp
##
## 0.7306201
##
## $sKp
##
## 0.2118354
##
## $ex.Kp
##
## -2.71838
##
## $teKp
##
## -0.7310657
##
##
## mmntSK> theoCentileSK("BCCG", mu=2, sigma=.2, nu=2)
## $IR
## [1] 0.5395878
##
## $SIR
## [1] 0.2697939
##
## $S_0.25
## [1] -0.06473032
##
## $S_0.01
## [1] -0.2317742
##
## $K_0.01
## [1] 3.746159
##
## $sK_0.01
## [1] 1.086143
##
##
## mmntSK> plotCentileSK(fam="BCCG",  mu=2, sigma=.2, nu=2)
```




```
##
## mmntSK> checkMomentSK(Y)
```




```
##
## mmntSK> checkCentileSK(Y)
```




```
##
## mmntSK> #checkCentileSK(Y, type="tail")
## mmntSK>
## mmntSK>
## mmntSK>
```

16 gamlss.demo()

```
#install.packages("gamlss.demo")
#Based on rpanel
library(gamlss.demo)

#The demo for gamlss distributions and smoothing
gamlss.demo()

#Examples
#t family distribution
demo.TF()

#Skew Normal Type 1 distribution
demo.SN1()

#Box-Cox Power Exponential distribution
demo.BCPE()

#Demos for smoothing techniques
demo.BSplines()
demo.PSplines()
demo.interpolateSmo()
```

```
demo.histSmo()

#Interface for demonstrating the gamlss.family distributions
demoDist()

#Demo for local polynomial smoothing
demoLpolyS()
```

17 Further details by using aids dataset

17.1 Likelihood function

A function to generate the likelihood function from a GAMLSS object

```
logLik_aids1<- gen.likelihood(aids1)
logLik_aids1(par=coef(aids1))
```

```
## [1] 203.1965
```

17.2 Variance-Covariance Matrix

```
vcov(aids1, type="all")
```

```
## $coef
##    (Intercept) pb(x, df = 10)
##     3.31103262     0.07004591
##
## $se
##    (Intercept) pb(x, df = 10)
##    0.037778910    0.001123696
##
## $vcov
##                   (Intercept) pb(x, df = 10)
## (Intercept)     0.0014272460  -4.077030e-05
## pb(x, df = 10) -0.0000407703   1.262694e-06
##
## $cor
##                 (Intercept) pb(x, df = 10)
## (Intercept)       1.0000000     -0.9603853
## pb(x, df = 10)  -0.9603853      1.0000000
```

17.3 Confidence Intervals

```
confint(aids1)
```

```
##                     2.5 %     97.5 %
## (Intercept)    3.2369873 3.38507792
## pb(x, df = 10) 0.0678435 0.07224831
```