Flexible Regression and Smoothing
The GAMLSS packages in R

Mikis Stasinopoulos, Bob Rigby,
Vlasios Voudouris, Gillian Heller
and **Fernanda De Bastiani**

July 23, 2015

Contents

List of Figures

List of Tables

Preface

Regression analysis is one of the most popular and powerful statistical techniques for exploring the relationship between a response variable and explanatory variables of interest. Like all models, regression models are based on assumptions which need to be true (or approximately true) for the model to have valid conclusions. Practitioners who use the standard linear regression model soon find that the classical assumptions about normality and constant variance of the errors terms and linearity of the relationship between response variable and the explanatory variables very seldom hold. Generalised Linear Models (GLM) and Generalised Additive models (GAM), were introduced by Nelder and Wedderburn [1972] and Hastie and Tibshirani [1990] respectively to overcome some of the limitations of the standard linear model. These days the GLM's and (to a less extent) the GAM's are textbook material and are very popular with practitioners.

Unfortunately, especially with larger data sets, those models are found to have inadequate fits or to be inappropriate in a lot of practical situations. In this book we are dealing with the Generalised Additive Models for Location, Scale and Shape, (GAMLSS), a framework which corrects some of the problems of GLM's and GAM's. A GAMLSS model is a general regression model which assumes that the response (dependent) variable has any parametric distribution. In addition all the parameters of the distribution of the response variable can be modelled as functions of the available explanatory variables. This is in contrast to GLM's and GAM's where the distribution of the response variable is restricted to the exponential family of distributions and only the mean (a location parameter) of the distributions is modelled. So the main characteristic of GAMLSS models is the ability to allow the location, scale and shape of the distribution of the response variable to vary according to the values of explanatory variables.

This is a book about Generalised Additive Models for Location, Scale and Shape, (GAMLSS), and its implementation in **R**. The GAMLSS model is implemented through a series of **R** packages.

The aim of the book is:

to introduce the basic ideas of the GAMLSS models,

to show how the models can be fitted in **R**,

to demonstrate the capabilities (and limitations) of **R** GAMLSS packages,

to provide a sufficiently wide range of examples in order to demonstrate the usefulness of the GAMLSS models,

to help practitioners to understand the ideas behind the GAMLSS models,

to provide information about the GAMLSS implementation in **R**.

This book is written for:

practitioners who wish to understand and use the GAMLSS models

students who wish to learn GAMLSS through practical examples and

for us the authors who often forget what we have done in the past and require documentation to remember it.

We assume that practitioners and students are familiar with the basic concepts of regression and have a minimum experience with **R**. All **R** commands are available within the text and the reader is encouraged to learn by actually repeating the examples given within the book. Matrix algebra is used for describing the models, so some knowledge of matrices will be useful.

This book is not designed to be read necessarily from the beginning to the end. What we are hoping to achieve is an easy going introduction to the GAMLSS models, and something which practitioners could refer to, describing the different functionalities of the GAMLSS **R** packages. With this in mind we divide the book in several district parts dealing with different aspects of the statistical 'regression type' of modelling:

part I Introduction to models and packages: This part provides an explanation of why GAMLSS models are needed and information about the GAMLSS **R** packages using two practical examples.

part II The R implementation, algorithms and functions: This part is designed to help users to familiarise with the GAMLSS algorithms as well as the few basic functions of the main `gamlss` package and the created `gamlss` **R** objects.

part III Distributions: This part describes the different available distributions for the response variable. They are the distributions available in the package `gamlss.dist` but also distributions which can be generated by transforming, truncating and finite mixing. The comprise continuous, discrete and mixed (i.e. continuous-discrete) distributions, which can be highly skewed (positively or negatively) ans/or highly platykurtotic or leptokurtotic (i.e. light or heavy tails).

part IV Additive terms: This part shows the different ways additive terms can be used within a GAMLSS model. In particular it explains linear and non-linear parametric terms and non-linear smoothing terms which can be used to explain how the different explanatory variables effect specific distribution parameter. This part also gives examples of other possible terms which can be used (for example neural networks).

part V Model selection and diagnostics: Model selection is crucial in statistical modelling. This part explains the different methods and tools within the GAMLSS packages model for model selection and diagnostics

part VI Applications: Some interesting applications of the GAMLSS models are shown in this part.

part VII Appendix GAMLSS reference card: this part shows all the available functions within the different GAMLSS **R** package for reference.

Notation used in this book

In this book we would like to district between statistical models, **R** packages and **R** functions. We will use capital letters for models, bold characters for packages and code type characters (with extra brackets) for functions. For example

- GAMLSS : refers to the statistical model,
- **gamlss** : refers to **R** package and
- `gamlss()` to the **R** function.

Vectors in general will be represented in a lower case bold letters, e.g. $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ while matrices in an upper case bold letter, for example $\mathbf{X}$. Scalar random variables are represented by upper case, for example Y. The observed value of a random variable is represented by lower case, for example y.

Tables 1 shows the notation that will be used throughout this book.

	Systematic part
Y:	a univariate response variable
$\mathbf{y}$:	the vector of observed values of the response variable, i.e. $(y_1, y_2, \ldots, y_n)^\top$
n :	total number of observations
K :	the total number of parameters in the distribution of Y
k :	a parameters number $k = 1, \ldots, K$
p_k :	the number of columns in the design matrix $\mathbf{X}_k$
J_k :	the total number of smoothers for the kth distribution parameter, $\boldsymbol{\theta}_k$
q_{kj} :	the dimension of the random effect vector $\boldsymbol{\gamma}_{kj}$
$\mathbf{x}_{kj}$:	the $j$the explanatory variable vector for the kth parameter, $\boldsymbol{\theta}_k$
$\mathbf{X}_k$:	an $n \times p_k$ fixed effects design matrix for the kth parameter, $\boldsymbol{\theta}_k$
$\boldsymbol{\beta}_k$:	a vector of fixed effect parameters for the kth parameter, $\boldsymbol{\theta}_k$, i.e. $(\beta_1, \beta_2, \ldots, \beta_{p_k})^\top$
$\boldsymbol{\gamma}_{kj}$:	the j random effect parameter vector for the kth parameter, $\boldsymbol{\theta}_k$, of length q_{kj}
$\mathbf{Z}_{kj}$:	an $n \times q_{kj}$ random effect design matrix for the jth smoother of the $k$the parameter, $\boldsymbol{\theta}_k$
$\mathbf{G}_{kj}$:	an $q_{kj} \times q_{kj}$ matrix of penalties for $\boldsymbol{\gamma}_{kj}$
$\boldsymbol{\eta}_k$:	the predictor for the kth distribution parameter i.e. $\boldsymbol{\eta}_k = g_k(\boldsymbol{\theta}_k)$
$\mathbf{H}_k$:	the hat matrix for the kth parameter
$\mathbf{z}_k$:	the adjusted dependent variable for the kth parameter
$g_k()$:	link function applied to model the kth distribution parameter
$s_{kj}()$:	the jth non-parametric or non-linear function (in the predictor $\boldsymbol{\eta}_k$)
$\mathbf{W}$:	a $n \times n$ diagonal matrix of weights
$\mathbf{w}$:	a n dimensional vector of weights (the diagonal elements of $\mathbf{W}$)
$\mathbf{S}_{kj}$:	the jth smoothing matrix for the kth parameter
	Distributions
$f()$:	theoretical probability (density) function of the random variable Y [1], (`d` function)
$D()$:	a general probability (density) function, equivalent to $f()$
$F()$:	cumulative distribution function of the random variable Y (`p` function)
$Q()$:	inverse cumulative distribution function of the random variable Y (`q` function), i.e. $F_Y^{-1}()$
$E()$:	Expectation of random variable Y

[1] Occasionally the subscript Y is added if more than one random variables are involved for clarification i.e. $f_Y()$.

$Var()$:	Variance of random variable Y
$f_{Y/X}()$:	conditional probability of the random variable Y given X
$\phi()$:	probability density function of a standard normal distribution
$\Phi()$:	cumulative probability density function of a standard normal distribution
$\pi()$:	prior probability density function in a finite mixtures
$\boldsymbol{\pi}$:	vector of prior (or mixing) probabilities in a finite mixtures $\boldsymbol{\pi} = (\pi_1, \pi_2 \ \ldots, \pi_k)^\top$
	Distributions parameters
θ_k :	the kth distribution parameter, where $\theta_1 = \mu$, $\theta_2 = \sigma$, $\theta_3 = \nu$ and $\theta_4 = \tau$
$\boldsymbol{\theta}_k$:	a vector of length n of the kth distribution parameter, e.g. $\boldsymbol{\theta}_2 = \boldsymbol{\sigma}$
$\boldsymbol{\theta}$:	the vector of all the parameters of the distribution, e.g. $\boldsymbol{\theta} = (\mu, \sigma, \nu, \tau)^\top$
μ :	the first parameter of the distribution (usually location)
σ :	the second parameter of the distribution (usually scale)
ν :	the third parameter of the distribution (usually shape, e.g. skewness)
τ :	the fourth parameter of the distribution (usually shape, e.g. kurtosis)
λ :	a hyper-parameter
$\boldsymbol{\lambda}$:	the vector of all hyper-parameters in the model
σ_b :	standard deviation of a normal random effect term for a parameter θ_k
Z :	standard normal random variable, $NO(0, 1)$
z :	standard normal (Gaussian) quadrature mass point
	Likelihood and information criteria
L :	likelihood function
ℓ :	log likelihood function
Λ :	generalized likelihood ratio test statistic
$\boldsymbol{i}()$:	Fisher's expected information matrix
$I()$:	observed information matrix
GD :	global deviance, i.e. minus twice the fitted log-likelihood
$GAIC$:	generalized Akaike information criterion [(]i.e. $GD + (k \times df)$]
df :	total (effective) degrees of freedom used in the model
k :	penalty for each degree of freedom in the model
	Residuals
$\mathbf{u}$:	vector of (randomised) quantile residuals
$\mathbf{r}$:	vector of normalised (randomised) quantile residuals
$\boldsymbol{\varepsilon}$:	vector of (partial) residuals
Q:	Q statistic calculated from the residuals
Z:	Z-statistic calculated from the residuals
	GAMLSS model components
$\mathcal{M}$:	a GAMLSS model containing $\{\mathcal{D}, \mathcal{G}, \mathcal{T}, \Lambda\}$
$\mathcal{D}$:	the specification of the distribution of the response variable
$\mathcal{G}$:	the different link functions, e.g. $g_k()$ where $g_k(\theta_k) = \eta_k$
$\mathcal{T}$:	the explanatory variables terms influencing the distribution of Y
Λ:	the specification of the smoothing parameters
	vector operators
$\bullet$:	the Hadamard element by element product i.e . let $\mathbf{y}^\top = (y_1, y_2, y_3)^\top$ and $\mathbf{x}^\top = (x_1, x_2, x_3)^\top$ then $(\mathbf{y} \bullet \mathbf{x})^\top = (y_1 x_1, y_2 x_2, y_3 x_3)$

Table 1: Notation for the random and systematic part of a model used

Part I

Introduction to models and packages

This part contains two Chapters. The first explains why the GAMLSS models are needed while the second one can be seen as an introduction to GAMLSS implementation in **R**. Chapter 1 assumes some previous knowledge on linear regression and generalised linear models, as well as their representation using matrix algebra. Minimal knowledge of **R** is assumed at this stage since all commands needed are displayed.

Chapter 1

Why GAMLSS?

This chapter shows the evolution of statistical modelling from the linear model (LM) through the generalised linear model, GLM, the generalised additive model, GAM, to the generalised additive models foe location scale and shape, GAMLSS. It provides an

1. a discussion on the historical evolution of GAMLSS through a simple example
2. an introduction to the the GAMLSS models in **R**,
3. the definition of a GAMLSS model.

This chapter is the starting point for using GAMLSS in **R**.

1.1 Introduction

This chapter serves as an introduction to generalised additive models for location, scale and shape (GAMLSS). It builds up the GAMLSS model using ideas from its predecessors, in particular, from the linear regression models, the generalised linear models and the generalised additive models. It uses a relative simple example, the Munich `rent` data, to demonstrate why someone needs to use GAMLSS.

1.2 The 1980's Munich rent data

The `rent` data come from a survey conducted in April 1993 by Infratest Sozialforschung, where a random sample of accommodation with new tenancy agreements or increases of rents within the last four years in Munich was selected including: i) single rooms, ii) small apartments, iii) flats, iv) two-family houses. The data were analysed extensively, by Fahrmeir *et al.* (1994, 1995), and they are in the package **gamlss.data** which is automatically loaded when the **gamlss** package is loaded using the command `library(gamlss)`. There are 1967 observations and 9 variables in the data set but for the purpose of demonstrating GAMLSS, we will use only the following variables:

`R` : the response variable which is the monthly net rent in DM, i.e. the monthly rent minus calculated or estimated utility cost.

`Fl` : the floor space in square meters

`A` : the year of construction

`H` : a two level factor indicating whether there is central heating, (0), or not, (1).

`loc` : a factor indicating whether the location is below average, (1), average, (2), or above average, (3).

Figure 1.1

```
PPP <- par(mfrow=c(2,2))
plot(R~Fl, data=rent, col=gray(0.7), pch =  15, cex = 0.5)
plot(R~A, data=rent, col=gray(0.7), pch =  15, cex = 0.5)
plot(R~H, data=rent, col=gray(0.7), pch =  15, cex = 0.5)
plot(R~loc, data=rent, col=gray(0.7), pch =  15, cex = 0.5)
par(PPP)
```

R code on page 28

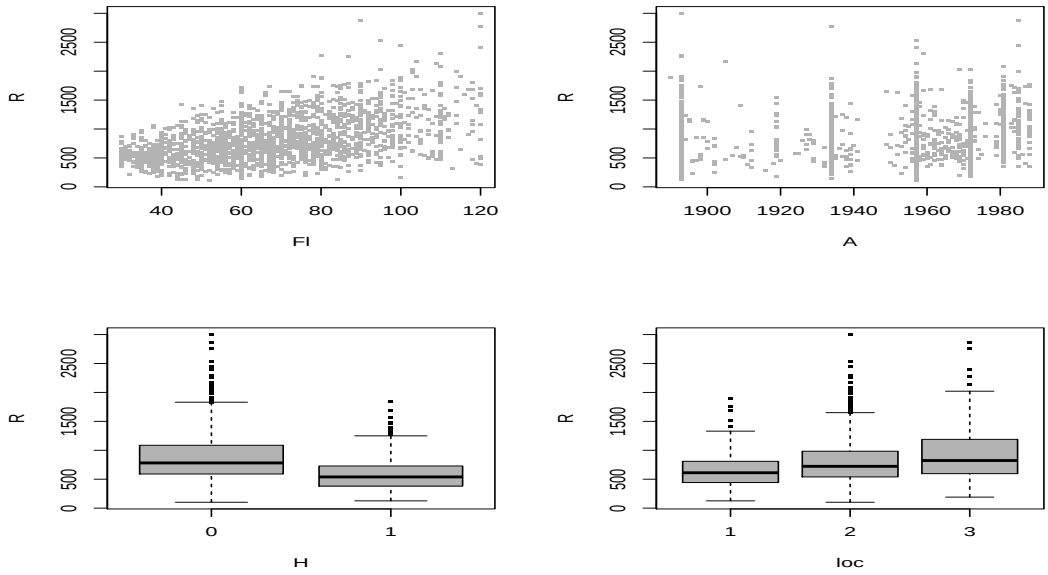


Figure 1.1: Plot of the rent `R` against explanatory variables `Fl`, `A`, `H` and `loc`.

Figure 1.1 shows plots of the net rent `R` against each of the above explanatory variables. Although these are bivariate exploratory plots and take no account of the interplay between the explanatory variables, they give an indication of the complexity of this data. The first two explanatory variables, `Fl` and `A`, are continuous. The plot of rent, `R`, against floor space, `Fl`, suggests a positive relationship with an increased variation for larger floor spaces. The assumption of homogeneity in the variance of the rent variable appears to be violated here. There is also some indication of positive skewness in the distribution of the rent variable. The peculiarity of the plot of rent, `R`, against year of construction, `A`, is due to the method of data collection.

Many of the observations of `A` were collected on an interval scale and assigned the value of the interval midpoint, while for the rest the actual year of construction was recorded. The plot suggests that for houses up to 1960 the median rent price is roughly constant but for flats constructed after that year there is an increasing trend in the median rent price. The remaining box and whisker plots display how the rent price varies according to the explanatory factors. The median rent price increases if the flat has central heating and increases as the location changes from below average to average and then to above average. There are no surprises in the plots here but again the problem of skewness is prominent with non symmetrical boxes about the median and longer upper than lower whiskers.

In summary, any statistical model used for the analysis of the above data should be able to deal with the following statistical problems:

Problem I : The complexity of the relationship between net rent and the explanatory variables. The dependence of the median of the response variable rent on floor space and age of construction is non-linear and non-parametric smoothing functions may be needed. Median rent may also depend on non-linear interactions between the explanatory variables.

Problem II : Non-homogeneity of variance of rent. There is clear indication of non-homogeneity of the variance of rent. The variance of the response variable rent may depend on its mean and/or explanatory variables e.g. floor space. A statistical model is needed where this dependence can be modelled explicitly.

Problem III : Skewness in the distribution of the response variable rent. There is clear indication of skewness in the distribution of net rent and this has to be accounted for within the statistical model.

1.3 The linear regression model

A simple but effective model, (which served the statistical community well for the main part of the last century), is the linear regression model

$$Y_i = \beta_0 + \beta_1 x_{1i} + \ldots + \beta_r x_{ri} + \epsilon_i \qquad \text{where } \epsilon_i \overset{\text{ind}}{\sim} N(0, \sigma^2) \tag{1.1}$$

for $i = 1, 2, \ldots, n$. It assumes that the error terms, ϵ_i for $i = 1, \ldots, n$, are independently distributed normal variables (that is the meaning of the notation $\overset{\text{ind}}{\sim}$) with zero mean and constant variance σ^2. This specification is equivalent to

$$\begin{aligned} Y_i &\overset{\text{ind}}{\sim} N(\mu_i, \sigma^2) \\ \mu_i &= \beta_0 + \beta_1 x_{1i} + \ldots + \beta_r x_{ri}, \end{aligned} \tag{1.2}$$

for $i = 1, 2, \ldots, n$. To avoid subscript problems later on we rewrite the model in (1.2) in matrix form as:

$$\begin{aligned} \mathbf{y} &\overset{\text{ind}}{\sim} N(\boldsymbol{\mu}, \sigma^2). \\ \boldsymbol{\mu} &= \mathbf{X}\boldsymbol{\beta} \end{aligned} \tag{1.3}$$

where $\mathbf{X}$ is an $n \times p$ design matrix ($p = r+1$) containing all the appropriate explanatory variable columns (plus a column of ones if the the constant is required) and $\boldsymbol{\beta}$ is the vector unknown vector of p coefficients to be estimated using the data. The notation $\mathbf{y} \overset{\text{ind}}{\sim} N(\boldsymbol{\mu}, \sigma^2)$, represents $Y_i \overset{\text{ind}}{\sim} N(\mu_i, \sigma^2)$ for $i = 1, \ldots, n$. Note that in order for the model to be fitted both $\boldsymbol{\beta}$ and σ^2 have to be estimated from the data. The usual practice is to estimate the $\boldsymbol{\beta}$ using the least square estimator

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} \tag{1.4}$$

which can be shown to be the maximum likelihood estimator, MLE, for $\boldsymbol{\beta}$.

Let $\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}}$ be the fitted values of the model and $\hat{\boldsymbol{\epsilon}} = \mathbf{y} - \hat{\mathbf{y}}$ be the standard residuals of the model. Then a maximum likelihood estimator for σ^2 is

$$\hat{\sigma}^2 = \frac{\hat{\boldsymbol{\epsilon}}^\top \hat{\boldsymbol{\epsilon}}}{n} \tag{1.5}$$

The MLE $\hat{\sigma}^2$ for is a biased estimator of σ^2 , i.e. $E\left[\hat{\sigma}^2\right] \neq \sigma^2$, so an unbiased estimator of σ^2 , given by

$$s^2 = \frac{\hat{\boldsymbol{\epsilon}}^\top \hat{\boldsymbol{\epsilon}}}{n-p} \tag{1.6}$$

with $E\left[s^2\right] = \sigma^2$, is often used instead, where p is the rank of the matrix $\mathbf{X}$. Sometimes the unbiased estimator in (1.6) is referred as the *REML* estimator of σ^2.

A linear regression model can be fitted in **R** using the function `lm()`. Here we compare the results from `lm()` to the ones obtained by the function `gamlss()` of the package **gamlss**.

```
r1 <-  gamlss(R ~ Fl+A+H+loc, family=NO,  data=rent)

## GAMLSS-RS iteration 1: Global Deviance = 28159
## GAMLSS-RS iteration 2: Global Deviance = 28159

l1 <- lm(R ~ Fl+A+H+loc,data=rent)
coef(r1)

##  (Intercept)           Fl            A           H1         loc2
## -2775.038803     8.839445     1.480755  -204.759562   134.052349
##         loc3
##   209.581472

coef(l1)

##  (Intercept)           Fl            A           H1         loc2
## -2775.038803     8.839445     1.480755  -204.759562   134.052349
##         loc3
##   209.581472
```

The fitted beta coefficients of the two fits are identical. Note that the two factors of the `rent` data, `H` and `loc` are fitted as dummy variables as explained in more detail in Section 8.2.1.

The fitted objects `r1` and `l1` use the methods `fitted()` and `resid()` to obtained the fitted values and the residuals respectively for their models. Note though that the `gamlss` object residuals are not simply the $\hat{\boldsymbol{\epsilon}} = \mathbf{y} - \hat{\mathbf{y}}$ residuals but are the normalised (randomised) quantile

residuals as explained in section 12.2 of Chapter 12. Randomisation happens only for discrete or interval response variables.

Important: GAMLSS uses normalised (randomised) quantile residuals.

The ML estimate of σ (not σ^2) can be obtained with `gamlss` using the command `fitted(r1, "sigma)[1]` while `summary()` will show the standard errors and t-test for the estimates.

```
fitted(r1, "sigma")[1]
```

```
##        1
## 308.4768
```

```
summary(r1)
```

```
## ******************************************************************
## Family:  c("NO", "Normal")
##
## Call:  gamlss(formula = R ~ Fl + A + H + loc, family = NO, data = rent)
##
## Fitting method: RS()
##
## ------------------------------------------------------------------
## Mu link function:  identity
## Mu Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2775.0388   470.1352  -5.903 4.20e-09 ***
## Fl              8.8394     0.3370  26.228  < 2e-16 ***
## A               1.4808     0.2385   6.208 6.55e-10 ***
## H1           -204.7596    18.9858 -10.785  < 2e-16 ***
## loc2          134.0523    25.1430   5.332 1.09e-07 ***
## loc3          209.5815    27.1286   7.725 1.76e-14 ***
## ---
## 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)  5.73165    0.01594   359.7   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## ------------------------------------------------------------------
## No. of observations in the fit:  1969
## Degrees of Freedom for the fit:  7
##       Residual Deg. of Freedom:  1962
##                       at cycle:  2
##
## Global Deviance:     28159
```

```
##             AIC:     28173
##             SBC:     28212.1
## *******************************************************************
```

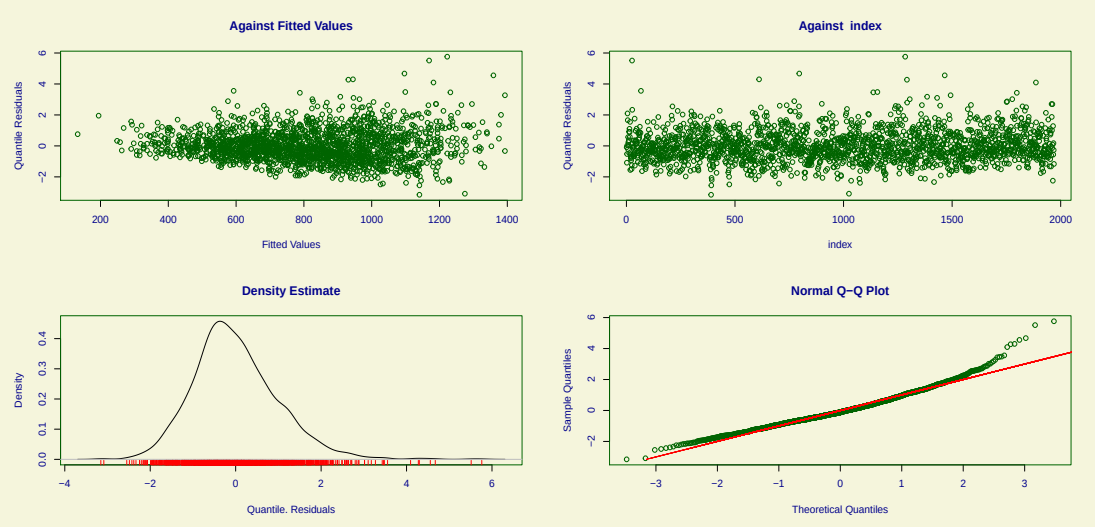
Note that σ is fitted in the log scale so in order to get its fitted value from the coefficient of σ we have to exponentiate, i.e. $\hat{\sigma} = \exp\left(\hat{\beta}_\sigma\right) = \exp(5.7316465) = 308.4767579$. Note that if you want R^2 from your `gamlss` output you can still get it using `Rsq(r1)`.

Figure 1.2

One way of checking the adequacy of your model is to look at the residuals.

```
plot(r1)
```

```
## *******************************************************************
##        Summary of the Quantile Residuals
##                             mean   =  4.959549e-13
##                         variance   =  1.000508
##                 coef. of skewness  =  0.7470097
##                 coef. of kurtosis  =  4.844416
## Filliben correlation coefficient   =  0.9859819
## *******************************************************************
```



R code on page 32

Figure 1.2: A residual plot of the linear model `r1`

More about the interpretation of the four plots in Figure 1.2 can be found in Section **??** of Chapter 12. But the important thing here is that distributional assumption that the data comes from a normal distribution is easily rejected by looking at the QQ-normal plot at the bottom right of Figure 1.2. We are moving next to the generalised linear model (GLM).

1.4 The generalised linear model (GLM)

The generalised linear model, (GLM), was introduced by Nelder and Wedderburn [1972] and further developed in McCullagh and Nelder [1989]. There are three major innovations in their approach: i) the normal distribution for the response variable $\mathbf{y}$ is replaced by the exponential family of distribution (denoted here as *ExpFamily*), ii) a monotonic *link* function $g(.)$ is used in modelling the relationship between μ_i and the explanatory variables and finally iii) in order to find the MLE for the beta parameters it uses an iteratively reweighed least squares algorithm which can be implemented easily on any statistical package having a good weighted least squares algorithm. The GLM model can be written as:

$$\begin{aligned} \mathbf{y} &\overset{\text{ind}}{\sim} ExpFamily(\boldsymbol{\mu}, \phi) \\ g(\boldsymbol{\mu}) &= \mathbf{X}\boldsymbol{\beta}. \end{aligned} \tag{1.7}$$

The exponential family distribution $ExpFamily(\mu, \phi)$ is defined by the probability (density) function $f_Y(y; \mu, \phi)$ of Y having the form:

$$f_Y(y; \mu, \sigma) = \exp\left\{\frac{y\theta - b(\theta)}{\phi} + c(y, \phi)\right\} \tag{1.8}$$

where $E(Y) = \mu = b^{'}(\theta)$ and $V(Y) = \phi V(\mu)$ where the *variance function* $V(\mu) = b^{''}[\theta(\mu)]$. The form of (1.8) includes many important distributions including the normal, Poisson, gamma, inverse Gaussian and Tweedie, (Tweedie, 1984), distributions having variance functions $V(\mu) = 1, \mu, \mu^2, \mu^3$ and μ^p for $p < 0$ or $p > 1$, respectively, and also the binomial with variance function $V(\mu) = \frac{\mu(1-\mu)}{N}$. Within the GLM framework the Gaussian distribution, used in the previous section to fit the `rent` data, can be replaced by a gamma or inverse Gaussian distribution. We first fit the gamma distribution specified, using the `family` argument, as `GA` and `Gamma` respectively for GAMLSS and GLM models using functions `gamlss` and `glm` respectively. GAMLSS uses a log link function as default for μ and σ, since the range of both parameters is $(0, \infty)$. Link functions are used, in general within the package **gamlss.dist**, (which is automatically loaded if the **gamlss** package is loaded), to ensure that the parameters of the distributions are within their proper range. All available distributions within the package **gamlss.dist** together with their appropriate link functions for their parameters are shown in Tables 6.1, 6.2, 6.3 of Chapter **??**. The `glm(()` function has as default the *canonical* link function for μ which is different for each distribution and for the `Gamma` is the "inverse".

Important: The GAMLSS model as implemented in the the package **gamlss** does not use canonical links as default for μ as in the `glm()` function but generally uses links reflecting the range of the parameter values, i.e. "identity" for $(-\infty, \infty)$, "log" for $(0, \infty)$, "logit" for $(0, 1)$ etc.

Next we fit a gamma distribution model with "log" link for μ using the `glm()` function in **R** and the `gamlss()` function. A "log" link assumes that the relationship between μ and the predictor variables is multiplicative.

```
l2 <- glm(R ~ Fl+A+H+loc, family=Gamma(link="log"),  data=rent)
r2 <-  gamlss(R ~ Fl+A+H+loc, family=GA,  data=rent)

## GAMLSS-RS iteration 1: Global Deviance = 27764.59
## GAMLSS-RS iteration 2: Global Deviance = 27764.59
```

```
coef(l2)
```

```
##  (Intercept)           Fl            A           H1         loc2
##  2.864943806  0.010623194  0.001510066 -0.300074001  0.190764594
##         loc3
##  0.264083376
```

```
deviance(l2)
```

```
## [1] 282.5747
```

```
coef(r2)
```

```
## (Intercept)          Fl           A          H1        loc2        loc3
##  2.86497701  0.01062319  0.00151005 -0.30007446  0.19076406  0.26408285
```

```
deviance(r2)
```

```
## [1] 27764.59
```

The fitted coefficients from the two models are identical, but their correspondent deviances are not because they are defined differently. The GML deviance is defined as

$$D_{GLM} = -2\log\left(\frac{\hat{L}_c}{\hat{L}_s}\right)$$

where $\hat{L}_c$ is the fitted likelihood of the current fitted model and $\hat{L}_s$ is the fitted likelihood of the *saturated* model (the model where in modelling μ a parameter is fitted for each observation, i.e. zero degrees of freedom left). The GAMLSS deviance is just

$$D_{GAMLSS} = -2\log\hat{L}_c$$

and we refer to it as the *global deviance* or GDEV.

To get the coefficients with their standard errors use:

```
summary(r2)
```

```
## ******************************************************************
## Family:  c("GA", "Gamma")
##
## Call:  gamlss(formula = R ~ Fl + A + H + loc, family = GA, data = rent)
##
## Fitting method: RS()
##
## ------------------------------------------------------------------
## Mu link function:  log
## Mu Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.8649770  0.5688561   5.036 5.18e-07 ***
## Fl           0.0106232  0.0004128  25.733  < 2e-16 ***
## A            0.0015100  0.0002890   5.226 1.92e-07 ***
## H1          -0.3000745  0.0231287 -12.974  < 2e-16 ***
## loc2         0.1907641  0.0305204   6.250 5.01e-10 ***
```

```
## loc3         0.2640828  0.0329211   8.022 1.78e-15 ***
## ---
## 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) -0.98220    0.01558  -63.05   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## ------------------------------------------------------------------
## No. of observations in the fit:  1969
## Degrees of Freedom for the fit:  7
##       Residual Deg. of Freedom:  1962
##                       at cycle:  2
##
## Global Deviance:     27764.59
##             AIC:     27778.59
##             SBC:     27817.69
## ******************************************************************
```

To check whether the normal, gamma or the inverse Gaussian distribution is better for the data compare the three models using the Generalised Akaike Criterion (GAIC):

```
r22 <-  gamlss(R ~ Fl+A+H+loc, family=IG,  data=rent)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 27991.56
## GAMLSS-RS iteration 2: Global Deviance = 27991.56
```

```
GAIC(r1, r2, r22) # AIC
```

```
##     df      AIC
## r2   7 27778.59
## r22  7 28005.56
## r1   7 28173.00
```

```
GAIC(r1, r2, r22, k=log(length(rent$R))) # SBC or BIC
```

```
##     df      AIC
## r2   7 27817.69
## r22  7 28044.66
## r1   7 28212.10
```

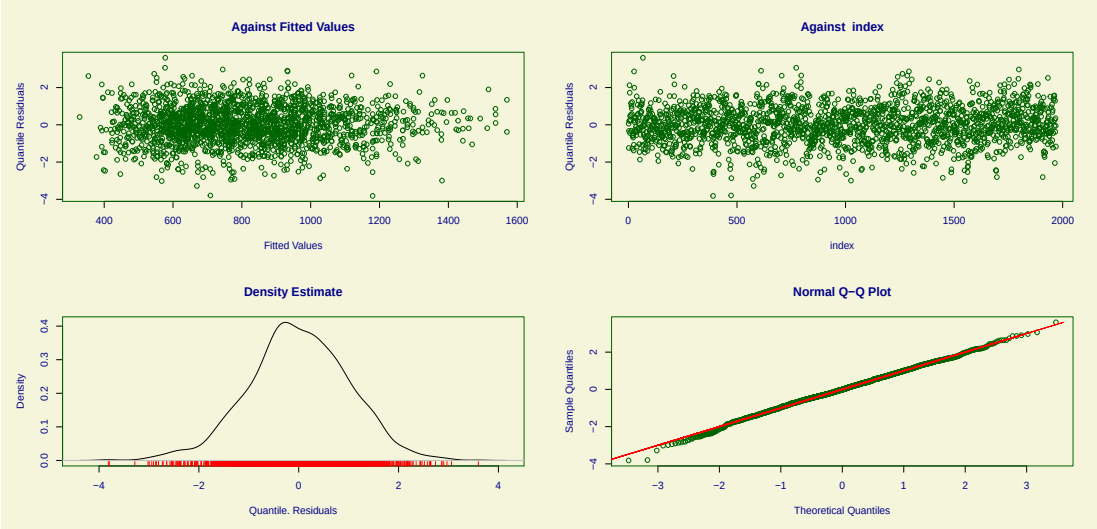
The conclusion is that according to both AIC or SBC the gamma fits better. no definition of GAIC is given yet Now we check the residuals:

Figure 1.3

```
plot(r2)
```

```
## ******************************************************************
##        Summary of the Quantile Residuals
##                             mean   =  0.0004795675
```

```
##                          variance   =  1.000657
##                 coef. of skewness   =  -0.1079453
##                 coef. of kurtosis   =  3.255464
## Filliben correlation coefficient   =  0.9990857
## ******************************************************************
```



R code on page 35

Figure 1.3: A residual plot of the generalised linear model `r2`

The residuals at this stage look a lot better than the normal distribution residuals of Figure 1.2 in that at least some of heterogeneity in the residuals has disappeared.

We next introduce the generalised additive model which allow more flexible modelling between the distribution parameter μ and the continuous explanatory variables.

1.5 The generalised additive model (GAM)

Smoothing techniques become popular in the late 1980's. Hastie and Tibshirani [1990] were the first to introduce them within the GLM framework and they give the name generalised additive models, GAM. Wood [2006] has contributed extensively to the GAM theory and popularity by allowing, in his implementation of GAM in **R** (package **mgcv**), the automatic calculation of the smoothing parameters in the model. (In the original implementation of GAM in S-plus and R the smoothing parameters λ or equivalently the effective degrees of freedom have to be fixed). The GAM model can be written as:

$$\mathbf{y} \overset{\text{ind}}{\sim} ExpFamily(\boldsymbol{\mu}, \phi) \tag{1.9}$$

$$g(\boldsymbol{\mu}) = \mathbf{X}\boldsymbol{\beta} + s_1(\mathbf{x}_1) + \ldots + s_J(\mathbf{x}_J) \tag{1.10}$$

where $s()$ stands for smoothing non-parametric functions applied to some of the continuous explanatory variables. The idea is to let the data determine the relationship between the linear

predictor $\eta = g(\mu)$ and the explanatory variables rather than enforcing a linear (or polynomial) relationship. Next chapter, Chapter 2, gives few examples of different smoothers. More detail about different smoothers within the **gamlss** package can be found in Chapter 9. Here we will use the smoothing function `pb()` which is an implementation of a P-splines smoother in GAMLSS, Eilers and Marx [1996]. Next we model the `rent` parameter μ using a smooth function for floor space `Fl` and age `A` and we compare the model using AIC with the simple GLM fitted in the previous section.

```
r3 <-  gamlss(R ~ pb(Fl)+pb(A)+H+loc, family=GA,  data=rent)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 27683.22
## GAMLSS-RS iteration 2: Global Deviance = 27683.22
## GAMLSS-RS iteration 3: Global Deviance = 27683.22
```

```
AIC(r2,r3)
```

```
##          df      AIC
## r3 11.21547 27705.65
## r2  7.00000 27778.59
```

According to the AIC the GAM model with smoothers is better than the simple GLM with only linear terms for `Fl` and age `A`. The summary of fit is shown below:

```
summary(r3)
```

```
## ******************************************************************
## Family:  c("GA", "Gamma")
##
## Call:
## gamlss(formula = R ~ pb(Fl) + pb(A) + H + loc, family = GA, data = rent)
##
## Fitting method: RS()
##
## ------------------------------------------------------------------
## Mu link function:  log
## Mu Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.0851197  0.5666171   5.445 5.84e-08 ***
## pb(Fl)       0.0103084  0.0004030  25.578  < 2e-16 ***
## pb(A)        0.0014062  0.0002879   4.884 1.12e-06 ***
## H1          -0.3008111  0.0225705 -13.328  < 2e-16 ***
## loc2         0.1886692  0.0299153   6.307 3.51e-10 ***
## loc3         0.2719856  0.0322699   8.428  < 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) -1.00196    0.01559  -64.27   <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:  1969
## Degrees of Freedom for the fit:  11.21547
##       Residual Deg. of Freedom:  1957.785
##                       at cycle:  3
##
## Global Deviance:     27683.22
##             AIC:     27705.65
##             SBC:     27768.29
## ******************************************************************
```

There is a "Note" on the output warning the users that because smoothers are fitted into the model the standard errors given should be treated with care. There are two issues associated with the output given by the `summary.gamlss()` function. The first is that the resulting coefficients of the smoothers and their standard errors refer only to the linear part of the smoother and not of the smoother's contribution as a whole which is decomposed into a linear plus a non-linear smoothing part. To test the contribution of the smoother as a whole (including the linear term) use the function `drop1()` as shown below. The second issue has to do with the standard errors of the linear part of the model that is of the terms `H` and `loc`. Those standard errors are estimated assuming that the smoother terms are fixed in their fitted values and therefore do not take into the account the uncertainty introduced by estimating the smoothing terms. Some suggestions for correcting this are given in Section **??**.

Important: When smoothers are fitted all standard errors shown should be treated with caution.

Now we use `drop1()` to check for the significance of the contribution of the smoothers (including the linear term).

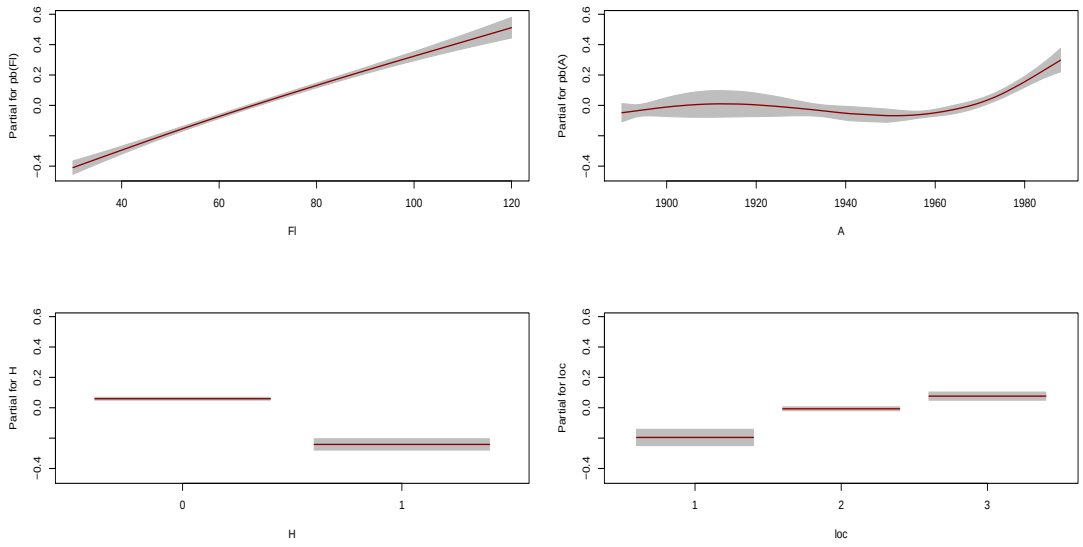
```
drop1(r3)
```

```
## Single term deletions for
## mu
##
## Model:
## R ~ pb(Fl) + pb(A) + H + loc
##            Df   AIC    LRT   Pr(Chi)
## <none>        27706
## pb(Fl) 1.4680 28261 558.59 < 2.2e-16 ***
## pb(A)  4.3149 27798 101.14 < 2.2e-16 ***
## H      1.8445 27862 160.39 < 2.2e-16 ***
## loc    2.0346 27770  68.02 1.825e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

All terms contributed significantly to modelling the predictor $\log \mu$. Note that `drop1()` can be very slow for large data sets and with a lot of smoother terms in the model. One of the properties of the fitted non-parametric smooth functions is that they can not simply be described in a mathematical form as for example parametric terms. However they can be displayed. Here we plot them using the the function `term.plot()`:

```
term.plot(r3, pages=1, ask=FALSE)
```

Figure 1.4



R code on page 39

Figure 1.4: A plot of the fitted terms for model `r3`

The plot shows that `rent` rises almost linearly with floor space `Fl`, but non-linearly with age `D` remaining stable if the flat was built before the 1960's and rising after that. For the two factors `H` and `loc` their contribution to rent is what we would expect, decrease if the flat does not have central heating (i.e. `H=`) and increasing as the location of the flat changes from below average to average and then to above average (i.e. `log`=,1,2 and 3 respectively). The shaded areas are the point-wise confidence bands for the smoothing curves and factor levels. The GAM models in general allow for a flexible specification of the dependence of the parameter predictors on different explanatory terms. To check the adequacy of the fitted GAM model we used a *worm plot* which is a de-trended QQ-plot of the residuals, van Buuren and Fredriks [2001].

```
wp(r3, ylim.all=.6)
```

Figure 1.5

Chapter 12 explain how to interpret a worm plot in detail. Here it is sufficient to say that for an adequate fitted model we would expect the dots (which appear like a little worm) to be close to the middle horizontal line and between the upper and lower dotted curves which act as 95% point wise confidence intervals. This does not appear to be the case for the fitted GAM model where the worm is well below of the lower curve in the left of the figure. Multiple worm plots

R code on page 39

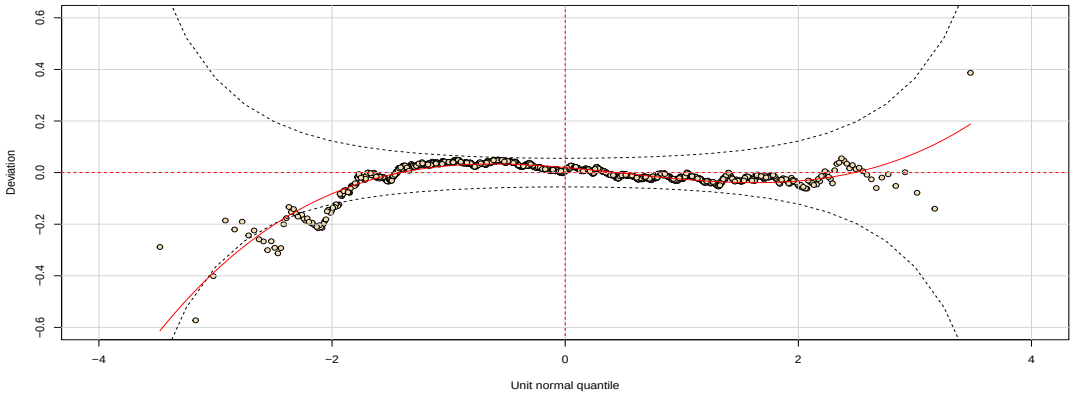


Figure 1.5: A plot of the fitted terms for model `r3`

allow investigation of the adequacy of the model within ranges of the explanatory variables. We shall next try to model the parameter σ of the Gamma distribution as a function of the explanatory variables.

1.6 Modelling the scale parameter

The gamma distribution has two parameters: i) μ which is the mean of the distribution and ii) σ which is a *scale* parameter and is related to the variance by the equation $V(Y) = \sigma^2\mu^2$. Up to now we have modelled only μ as a function of explanatory variables, but there are occasions (as for the Munich rent data) in which the assumption of a constant scale parameter is not appropriate. On those occasions modelling σ as a function of explanatory variables could solve the problem. Modelling σ started in the 1970-1980's. Harvey [1976] and Aitkin [1987] were the first to model the variance of the normal distribution as a function of explanatory variables. Engle [1982, 1995] was the first to propose a time series model for σ (volatility) for financial data, trying to solve the problem of heteroscedasticity. The model, which he called the ARCH (Autoregressive Conditional Heteroscedastic) model has created a whole industry of related models in finance. Modelling the dispersion parameter, $\phi = \sigma^2$, within GLM was done by Nelder and Pregibon [1987], Smyth [1989] and Verbyla [1993]. Rigby and Stasinopoulos [1996a,b] introduced smooth function for modelling both μ and σ and they call the mean and dispersion additive model (MADAM). In the original MADAM formulation the distribution has to be in the exponential family but the mode of fitting was Quasi-likelihood rather than full maximum likelihood used in GAMLSS.

Here we consider the following model:

$$\begin{aligned} \mathbf{y} &\overset{\text{ind}}{\sim} D(\boldsymbol{\mu}, \boldsymbol{\sigma}) \\ g_1(\boldsymbol{\mu}) &= \mathbf{X}_1\boldsymbol{\beta}_1 + s_{11}(\mathbf{x}_{11}) + \ldots + s_{1J_1}(\mathbf{x}_{1J_1}) \\ g_2(\boldsymbol{\sigma}) &= \mathbf{X}_2\boldsymbol{\beta}_2 + s_{21}(\mathbf{x}_{21}) + \ldots + s_{2J_2}(\mathbf{x}_{2J_2}) \end{aligned} \tag{1.11}$$

where $D(\boldsymbol{\mu}, \boldsymbol{\sigma})$ denotes any two parameter distribution in which both μ and σ are linear/smooth functions of the explanatory variables. Next we model the Munich rent data using the Gamma and the inverse Gaussian distributions in model (1.11):

```
r4 <-  gamlss(R ~ pb(Fl)+pb(A)+H+loc, sigma.fo=~pb(Fl)+pb(A)+H+loc, family=GA,
              data=rent)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 27572.14
## GAMLSS-RS iteration 2: Global Deviance = 27570.29
## GAMLSS-RS iteration 3: Global Deviance = 27570.28
## GAMLSS-RS iteration 4: Global Deviance = 27570.28
```

```
r5 <-  gamlss(R ~ pb(Fl)+pb(A)+H+loc, sigma.fo=~pb(Fl)+pb(A)+H+loc, family=IG,
              data=rent)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 27675.74
## GAMLSS-RS iteration 2: Global Deviance = 27672.97
## GAMLSS-RS iteration 3: Global Deviance = 27673
## GAMLSS-RS iteration 4: Global Deviance = 27673.01
## GAMLSS-RS iteration 5: Global Deviance = 27673.01
## GAMLSS-RS iteration 6: Global Deviance = 27673.02
```

```
AIC(r3, r4, r5)
```

```
##          df      AIC
## r4 22.25035 27614.78
## r3 11.21547 27705.65
## r5 21.82318 27716.66
```

The default link function for μ and σ in the gamma distribution (`GA`) in **gamlss** are $g_1(\mu) = \log \mu$ and $g_2(\sigma) = \log \sigma$. The model for the predictor for μ (i.e. $\log \mu$) is specified after `R~`, while the model for the predictor for the parameter σ (i.e. $\log \sigma$) is specified after `sigma.fo=~`. It is clear that the gamma distribution fits better than the inverse Gaussian as far as the AIC is concerns. To plot the fitted terms for σ use:

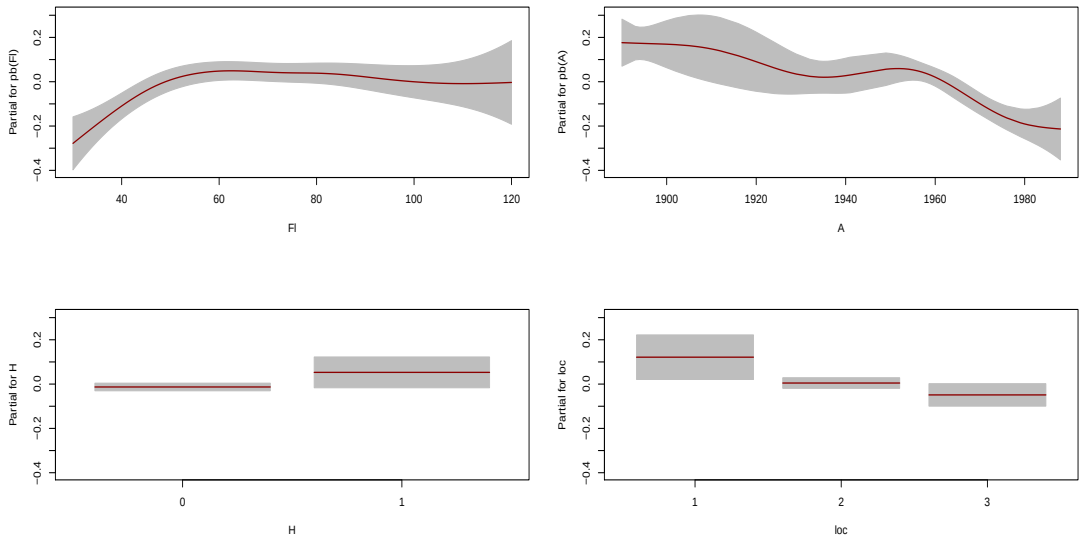
Figure 1.6

```
term.plot(r4, pages=1, what="sigma", ask=FALSE)
```

The significance of the terms can be tested using the `drop1()` function,

```
drop1(r4, what="sigma")
```

```
## Single term deletions for
## sigma
##
## Model:
## ~pb(Fl) + pb(A) + H + loc
##             Df   AIC    LRT   Pr(Chi)
## <none>         27615
## pb(Fl) 4.02694 27631 24.683 5.997e-05 ***
## pb(A)  3.87807 27659 52.167 1.067e-10 ***
## H      0.88335 27615  1.866   0.14788
## loc    2.03694 27619  8.036   0.01872 *
## ---
```



R code on page 41

Figure 1.6: A plot of the fitted terms for σ for model `r4`

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Every term apart from `H` seems to contribute significantly to explaining the behaviour of the σ parameter. To check the adequacy of the distribution use the `wp()` function.

Figure 1.7

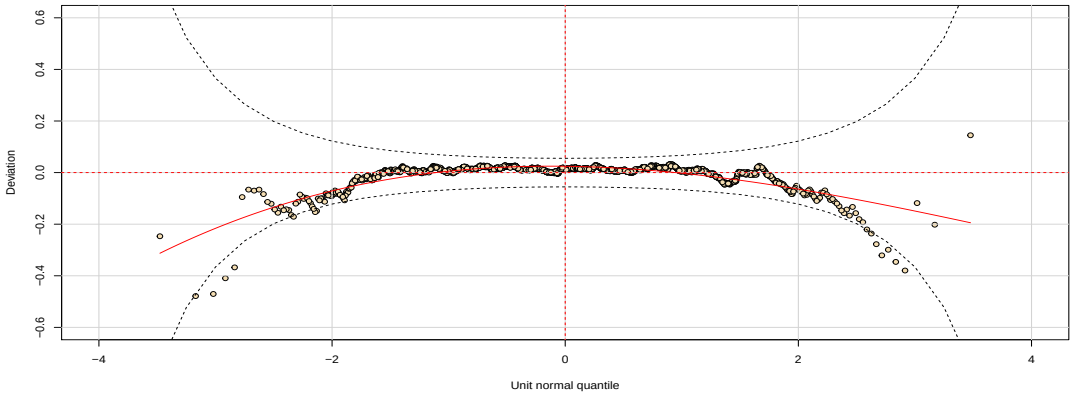
```
wp(r4, ylim.all=.6)
```

There are a few points of the worm plot falling outside 95% point-wise confidence intervals, indicating that the distribution may be inadequate.

1.7 The generalised additive model for location shape and scale.

One of the problems of a two parameter distribution is the fact that the skewness and kurtosis of the distribution are fixed for fixed μ and σ. With a relatively large set of data we would like to have the option of a more flexible skewness or kurtosis model. In this cases the model in (1.11) can be extended as follows:

$$\begin{aligned} \mathbf{y} &\overset{\text{ind}}{\sim} D(\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\nu}, \boldsymbol{\tau}) \\ g_1(\boldsymbol{\mu}) &= \mathbf{X}_1\boldsymbol{\beta}_1 + s_{11}(\mathbf{x}_{11}) + \ldots + s_{1J_1}(\mathbf{x}_{1J_1}) \\ g_2(\boldsymbol{\sigma}) &= \mathbf{X}_2\boldsymbol{\beta}_2 + s_{21}(\mathbf{x}_{21}) + \ldots + s_{2J_2}(\mathbf{x}_{2J_2}) \\ g_3(\boldsymbol{\nu}) &= \mathbf{X}_3\boldsymbol{\beta}_3 + s_{31}(\mathbf{x}_{31}) + \ldots + s_{3J_3}(\mathbf{x}_{3J_3}) \\ g_4(\boldsymbol{\tau}) &= \mathbf{X}_4\boldsymbol{\beta}_4 + s_{41}(\mathbf{x}_{41}) + \ldots + s_{4J_4}(\mathbf{x}_{4J_4}) \end{aligned} \tag{1.12}$$



R code on page 42

Figure 1.7: A plot of the fitted terms for model `r4`

where now $D(\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\nu}, \boldsymbol{\tau})$ is a four parameter distribution and where ν and τ are shape parameters of the distribution which are often related to the skewness and the kurtosis aspects of the distribution. Equation (1.12) defines the generalised additive model for location scale and shape (GAMLSS) first introduced by Rigby and Stasinopoulos [2005] and the main subject of this book. This book is about the GAMLSS implementation in **R**. The following comments related to model (1.12) are appropriate here:

Distributions The form of the distribution $D(\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\nu}, \boldsymbol{\tau})$ is general and only implies that the distribution should be in parametric form. In the current implementation there are around 100 *discrete*, *continuous* and *mixed* distributions implemented as `gamlss.family` including highly skew and kurtotic distributions. In addition:

- creating a *new* distribution is relatively easy see section **??**,
- any distribution in `gamlss.family` can be left, right or both sides *truncated*,
- a *censored* version of any `gamlss.family` distribution can be created allowing modelling of censored and interval response variables,
- any distribution in `gamlss.family` can be mixed to create a new finite mixture distribution as described in Chapter **??**,
- *Discretised* continuous distributions can be created for modelling discrete response variables see for example **??**.
- Any continuous `gamlss.family` distribution in $(-\infty, \infty)$ can be transformed to a distribution in $(0, \infty)$ or $(0, 1)$ using the arguments `type` with options `log` or `logit` respectively of the function `gen.Family()`.

Additive terms Explanatory variables can effect the parameters of the specified distribution in different ways. GAMLSS models allow this to be a linear or a non-linear parametric function or non-parametric smoothing functions. The **gamlss** package allows the following smooth additive terms: i) P-splines (Penalised B-splines) ii) monotone P-splines iii) cycle

P-splines iv) varying coefficient P-splines v) cubic smoothing splines vi) loess curve fitting vii) fractional polynomials viii) random effects ix) ridge regression and x) non-linear parametric fits. In addition through appropriate interfaces the software allows fitting of i) neural networks, via package **nnet** ii) decision trees, via package **rpar()** iii) random effects, via package **nlme**, iv) two dimensional smoothers, via package **mgcv**.

Fitting methods and Algorithms A parametric GAMLSS model [i.e. (1.12) without smoothing functions] is fitted by maximum likelihood estimation. The more general model is generally fitted my maximum penalised likelihood estimation. Chapter ???? shows that most of the smoothers used within GAMLSS can be written as $\mathbf{s}(\mathbf{x}) = \mathbf{Z}\boldsymbol{\gamma}$ where $\mathbf{Z}$ is a basis matrix depending on values of $\mathbf{x}$, and $\boldsymbol{\gamma}$ is a set of coefficients satisfying the quadratic penalty $\lambda\boldsymbol{\gamma}^\top\mathbf{G}\boldsymbol{\gamma}$ where λ is a smoothing parameter. Rigby and Stasinopoulos [2005] have shown that the algorithm used for fitting the GAMLSS model for fixed values of the smoothing parameters λ_{jk} is maximising a penalized likelihood function ℓ_p given by

$$\ell_p = \ell - \frac{1}{2}\sum_{k=1}^{4}\sum_{j=1}^{J_k}\lambda_{kj}\boldsymbol{\gamma}_{kj}^\top\mathbf{G}_{kj}\boldsymbol{\gamma}_{kj} \tag{1.13}$$

where $\ell = \sum_{i=1}^{n}\log f(y_i|\mu_i,\sigma_i,\nu_i,\tau_i)$ is the log likelihood function. Rigby and Stasinopoulos [2005] suggested two basic algorithms for fitting GAMLSS model (1.12). The first, the `CG` algorithm, is a generalization of the Cole and Green [1992] algorithm and uses the first derivatives and the (exact or approximate) expected values of the second and cross derivatives of the likelihood function with respect to $\boldsymbol{\theta} = (\mu,\sigma,\nu,\tau)$. However for many population probability (density) functions $f(y_i|\mu_i,\sigma_i,\nu_i,\tau_i)$ the parameters are information orthogonal (since the expected values of the cross derivatives of the likelihood function are zero), e.g. location and scale models and dispersion family models, or approximately so. In this case the second, the `RS` algorithm, which is a generalization of the algorithm used by Rigby and Stasinopoulos [1996a,b] for fitting the MADAM models, is more suited. (The `RS` algorithm does not use the expected values of the cross derivatives.)

We now return to the Munich data to see if we can improve the model by fitting a three parameter distribution. We will use here the BCCG distribution which is based on Cole and Green [1992] who were the first to fit a single smoothing term to each of the three parameters of the distribution. They called their model "the LMS method" and it is widely used for centile estimation, see Chapter **??**. The first model fits a constant ν while the second fits the same model for ν as was fitted for μ and σ.

```
r6 <-  gamlss(R ~ pb(Fl)+pb(A)+H+loc, sigma.fo=~pb(Fl)+pb(A)+H+loc,
              nu.fo=~1, family=BCCGo,  data=rent)

## GAMLSS-RS iteration 1: Global Deviance = 27628.31
## GAMLSS-RS iteration 2: Global Deviance = 27568.64
## GAMLSS-RS iteration 3: Global Deviance = 27566.3
## GAMLSS-RS iteration 4: Global Deviance = 27566.09
## GAMLSS-RS iteration 5: Global Deviance = 27566.06
## GAMLSS-RS iteration 6: Global Deviance = 27566.06
## GAMLSS-RS iteration 7: Global Deviance = 27566.06
## GAMLSS-RS iteration 8: Global Deviance = 27566.06

r7 <-  gamlss(R ~ pb(Fl)+pb(A)+H+loc,sigma.fo=~pb(Fl)+pb(A)+H+loc,
              nu.fo=~pb(Fl)+pb(A)+H+loc, family=BCCGo,  data=rent)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 27616.6
## GAMLSS-RS iteration 2: Global Deviance = 27553.71
## GAMLSS-RS iteration 3: Global Deviance = 27551.5
## GAMLSS-RS iteration 4: Global Deviance = 27551.32
## GAMLSS-RS iteration 5: Global Deviance = 27551.32
## GAMLSS-RS iteration 6: Global Deviance = 27551.32
## GAMLSS-RS iteration 7: Global Deviance = 27551.32
```

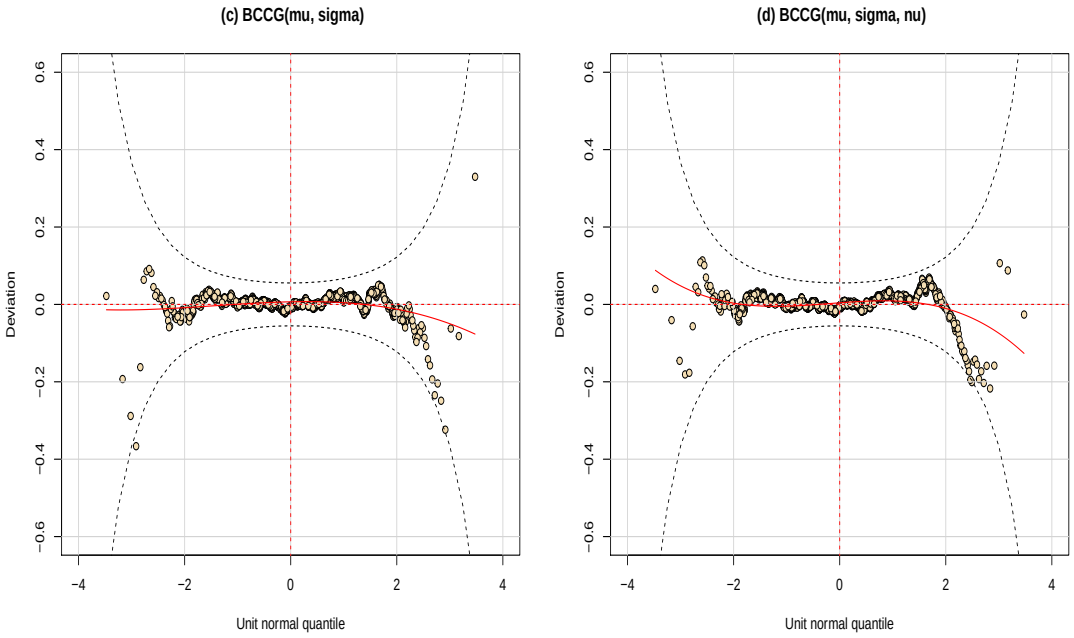
```
AIC(r4, r6, r7)
```

```
##           df      AIC
## r7 28.41391 27608.15
## r6 22.48092 27611.02
## r4 22.25035 27614.78
```

It look that the BCCG distribution provides a superior fit compared to the gamma distribution and that modelling the ν parameters as a function of the explanatory variables improves the fit. To check the adequacy of the fitted distribution we use the worm plots.

Figure 1.8

```
op <- par(mfrow=c(1,2))
wp(r6, ylim.all=.6) ;  title(" (c) BCCG(mu, sigma)")
wp(r7, ylim.all=.6); title(" (d) BCCG(mu, sigma, nu)")
par(op)
```



R code on page 45

Figure 1.8: A residual plot of the linear model `r1`

Both worm plots show an adequate fit, so we finish our demonstration here.

We have used the Munich rent data to demonstrate how GAMLSS can be used to model the data and we arrived at a more sophisticated model than using only GLM or GAM. In particular modelling both μ and σ parameter of a gamma $GA(\mu, \sigma)$ distribution in model `r4` provide a a substantial improved fit to the rent response variable as compared to the GAM model `r3`. Also a three parameter distribution model using the $BCCG(\mu, \sigma, \nu)$ distribution improves also the fit. GAMLSS provides greater flexibility in modelling regression type model but with this flexibility comes more responsibility for the statistician. This is not a bad thing. The philosophy of GAMLSS is to allow the practitioner to have a wider choice when trying to fit adequately a response variable.

We conclude this Chapter with some of the basic properties of GAMLSS:

- GAMLSS is a very flexible unifying framework for univariate regression type models.
- It allows *any* distribution for the response variable where *all* the parameters of the distribution can be modelled as a functions of explanatory variables.
- It allows a variety of (penalised) additive terms in the models for the distribution parameters.
- The fitted algorithm is modular, where different components can be added easily.
- It extends basic statistical models allowing flexible modelling of over-dispersion, excess of zeros, skewness and kurtosis in the data.

Chapter 2

Introduction to the gamlss packages

This chapter provides:

1. an introduction to GAMLSS package in **R**,
2. an introduction to some of the facilities of the **gamlss** packages using a simple regression model (with one explanatory variable).

2.1 Introduction

This Chapter uses a simple example of a continuous response variable against a continuous explanatory variable to demonstrate some of the facilities that the **R gamlss** packages provide. Section 2.2 describes the different GAMLSS packages in **R**. Section **??** provides a basic introduction of the **gamlss** package. Chapter 2.3 shows demonstrate the `gamlss()` function and other facilities in the **gamlss** package.

2.2 The GAMLSS packages

The GAMLSS framework comprise of several different packages written in the free software **R**, i.e. the original **gamlss** package and other add-on packages, i.e.

1. The original **gamlss** package for fitting a GAMLSS model. This packages depends on the **gamlss.dist** and **gamlss.data** packages. It contains the main function `gamlss()` for fitting a GAMLSS model and methods for dealing with fitted `gamlss` objects. Chapter 3 describes the algorithms use by the function `gamlss()`. Chapter 4 describes the arguments and how the `gamlss()` function can be used. Chapter 5 describes the different methods, (**R** functions), available for manipulating `gamlss` fitted objects.

2. The **gamlss.add** package for fitting extra additive terms. This package provides extra additive terms for fitting a parameter of the distribution of the response variable. This is mainly achieved by providing interfaces with other **R** packages. For example, neural networks, decision trees and multidimensional smoothers can be fitted within `gamlss()` by using the packages **nnet**, **rpart** and **mgcv** respectively. The use of those terms is explained in Chapter **??**,

3. The **gamlss.cens** package for fitting censored (left, right or interval) response variables. This package generates `gamlss.family` distributions suitable for fitting censored data within a GAMLSS model. By censoring we mean that the response variable is an interval response variable, that is, when the exact value of the event is not observed but only an interval of when the event happens. (An example is needed.)

4. The **gamlss.data** package for data used in this book. This package is automatically loaded if the **gamlss** package is loaded.

5. The **gamlss.demo** package for teaching purpose demos. The purpose of this package is twofold. Firstly, it provides a visual presentation to all `gamlss.family` distributions. That is, the user can visualise how the shape of the distribution is changing when any of the parameters of the distributions are changing. Secondly, it provides a visual presentation of some of the smoothing and P-splines ideas. Smoothing terms are used within a GAMLSS model to explored non linearities in the data.

6. The **gamlss.dist** package for `gamlss.family` distributions. This package contains all the distributions available in GAMLSS models and it is automatically loaded if the **gamlss** package is loaded. More information about the distribution available can be found in Chapter 6 and in the book *Distributions for Location Scale and Shape*.

7. The **gamlss.mx** package for fitting finite mixture distributions and non parametric random effects. This package provides two main functions: i) `gamlssMX()` for fitting finite mixture distributions appropriate for multimodal data and ii) `gamlssPN()` for fitting non-parametric random effects. The later function also provides a way of fitting finite mixtures when some of the parameters of the distributions are common within the mixtures. Chapter **??** provides examples for fitting finite mixtures to data while Chapter **??** provides information for fitting non-parametric random effects.

8. The **gamlss.nl** package for fitting non-linear parametric models within the GAMLSS framework. Chapter **??** provides information how to do that.

9. The **gamlss.spatial** package for spatial models. This package provides facilities so Markov Random Fields (MRF) terms can be fitted within a GAMLSS models. MRF are appropriate when a factor in the data provides a geographical information, for example areas in a region, and when we want to take the neighbourhood information into about when we build a model. (An example is needed)

10. The **gamlss.tr** package for fitting truncated distributions. This package can take any `gamlss.family` distribution and truncated it, (left, right or both), so it can be used as a response variable distribution within a GAMLSS models. (An example is needed)

The **R** and the GAMLSS framework packages can be downloaded and installed from CRAN, the **R** library at `http://www.r-project.org/`.

Help files are provided for all functions in the **gamlss** package in the usual way. For example using

```
help(package="gamlss")
?gamlss
```

will bring you information for the package **gamlss** and the function `gamlss()` respectively.

2.3 A simple example using the gamlss packages

The function `gamlss()` of the package **gamlss** is similar to the `gam()` functions in the **R** packages **gam**, Hastie [2006], and **mgcv**, Wood [2006], respectively 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. The function `gamlss()` also can be used to fit models which can be fitted using the functions `glm()` of **R** and parametric models `gamlss` and `glm()` should give identical results as far as the fitted values and the fitted coefficients for the mean are concerned (given that the same distribution from the exponential family is fitted). However for generalised linear models, the dispersion parameters ϕ is fitted by a moment estimator, while the `gamlss` scale parameter $\sigma = \phi^{1/2}$ is fitted by maximum likelihood estimation. For smoothing models the `gamlss` fitted mean model results should be identical to the `gam()` results of package **gam**, if the **gamlss** additive cubic spline function `cs()` is used and for fixed degrees of freedom, although note that the degrees of freedom specified by the user in the function `cs()` in the package **gamlss** are on the top of the constant and linear terms, while in `gam` the degrees of freedom are on top of the constant. Also the convergence criterion may need to be reduced for proper convergence in `gam()`. For smoothing models where the additive **gamlss** function `pb()` is used, `gamlss()` and `gam()` of package **mgcv** should produce similar but not necessarily identical results, if the same method of estimating the smoothing parameter is used. Note, however, that by default `pb()` in `gamlss()` uses local maximum likelihood estimation of the smoothing parameter, while `gam()` by default uses generalized cross-validation.

This implementation of `gamlss()` allows modelling of up to four parameters in a distribution family, which are conventionally called `mu`, `sigma`, `nu` and `tau`. Here we will try to give a simple demonstration of the **gamlss** package.

Data summary:
R data file: `film90` in package **gamlss.data** of dimensions 4015×14 but only two variables are used here.
variables
`lborev1` : the log of box office revenues after the first week calculated in 1987 prices (the response variable)
`lboopen` : the log of box office opening week revenues calculated in 1987 prices
purpose: to demonstrate the fitting of a simple regression type model in the **gamlss** package.

The data are analysed in Voudouris et al. [2012] where more information about the data and the purpose of the original analysis can be found. We use the data here for demonstrating some of the features of GAMLSS. The data contain several variables, but here we restrict only to two.

Figure 2.1

```
library(gamlss)
data(film90)
plot(lborev1~lboopen, data=film90, col="blue", xlab="log opening revenue",
     ylab="log extra revenue")
```

R code on page 50

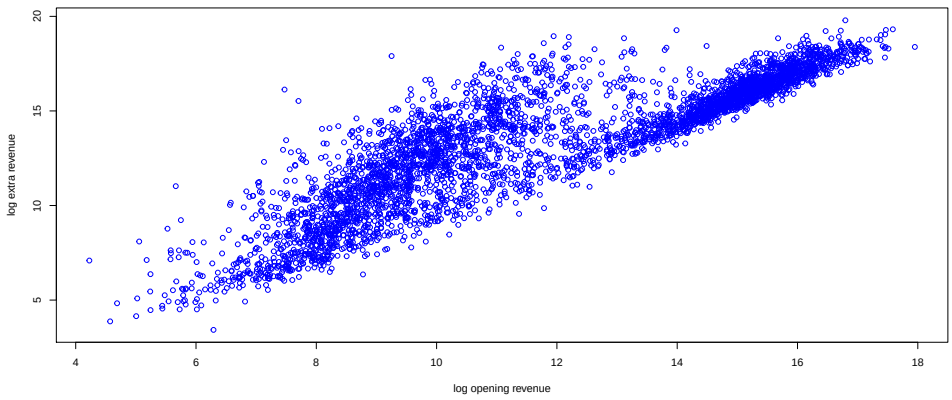


Figure 2.1: A plot of the `film90` revenues

The data are plotted in Figure 2.1.

2.3.1 Fitting a parametric model

First a simple linear regression model with normal errors is fitted to the data but it becomes obvious from Figure 2.2 that such a model does not fit well.

```
 m0 <- gamlss(lborev1~lboopen, data=film90)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 15078.88
## GAMLSS-RS iteration 2: Global Deviance = 15078.88
```

```
plot(lborev1~lboopen, data=film90,  col = "lightgray", lty=4)
lines(fitted(m0)~film90$lboopen)
```

Next a normal distribution is fitted with the mean of Y modelled as a cubic polynomial in x, i.e. `poly(x,3)`:

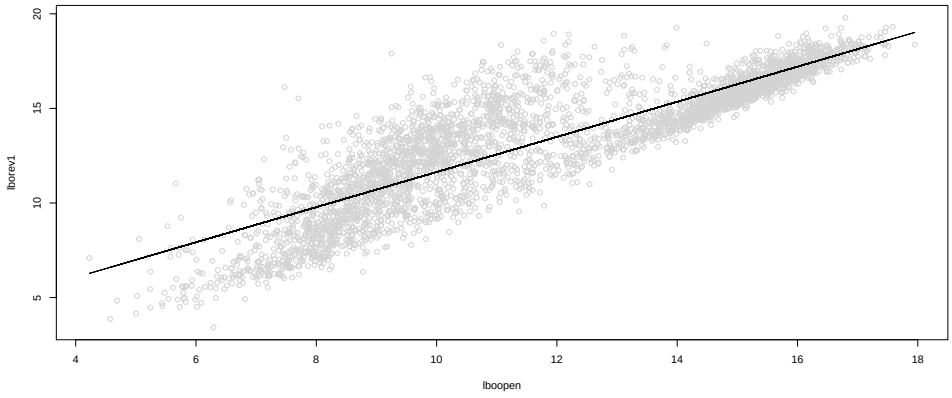
```
  m0 <- gamlss(lborev1~poly(lboopen,3), data=film90, family=NO)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 14517.65
## GAMLSS-RS iteration 2: Global Deviance = 14517.65
```

Since the normal distribution `NO` is also the default value we could omit the `family` argument. To get a summary of the results use:

```
summary(m0)
```

```
## ******************************************************************
```



R code on page 50

Figure 2.2: A plot of the `film90` data together with the fitted linear model for the mean

```
## Family:  c("NO", "Normal")
##
## Call:
## gamlss(formula = lborev1 ~ poly(lboopen, 3), family = NO, data = film90)
##
## Fitting method: RS()
##
## ------------------------------------------------------------------
## Mu link function:  identity
## Mu Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)        13.29257    0.02307  576.10   <2e-16 ***
## poly(lboopen, 3)1 180.61569    1.46494  123.29   <2e-16 ***
## poly(lboopen, 3)2 -29.66307    1.46494  -20.25   <2e-16 ***
## poly(lboopen, 3)3  20.30788    1.46494   13.86   <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)  0.38181    0.01114   34.28   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## ------------------------------------------------------------------
## No. of observations in the fit:  4031
## Degrees of Freedom for the fit:  5
##       Residual Deg. of Freedom:  4026
```

```
##                       at cycle:  2
##
## Global Deviance:     14517.65
##              AIC:     14527.65
##              SBC:     14559.15
## ******************************************************************
```

The **R** function `poly()` is used to fit orthogonal polynomials (see section 8.3), but we could have fitted the same model using the `I()` function, i.e.

```
m00 <- gamlss(lborev1~lboopen+I(lboopen^2)+I(lboopen^3), data=film90,
                       family=NO)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 14517.65
## GAMLSS-RS iteration 2: Global Deviance = 14517.65
```

```
summary(m00)
```

```
## ******************************************************************
## Family:  c("NO", "Normal")
##
## Call:  gamlss(formula = lborev1 ~ lboopen + I(lboopen^2) + I(lboopen^3),
##     family = NO, data = film90)
##
## Fitting method: RS()
##
## ------------------------------------------------------------------
## Mu link function:  identity
## Mu Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -2.244e+01  1.278e+00  -17.55   <2e-16 ***
## lboopen       7.174e+00  3.537e-01   20.28   <2e-16 ***
## I(lboopen^2) -4.985e-01  3.171e-02  -15.72   <2e-16 ***
## I(lboopen^3)  1.275e-02  9.195e-04   13.86   <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)  0.38181    0.01114   34.28   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## ------------------------------------------------------------------
## No. of observations in the fit:  4031
## Degrees of Freedom for the fit:  5
##       Residual Deg. of Freedom:  4026
##                       at cycle:  2
```

```
##
## Global Deviance:     14517.65
##              AIC:     14527.65
##              SBC:     14559.15
## ******************************************************************
```

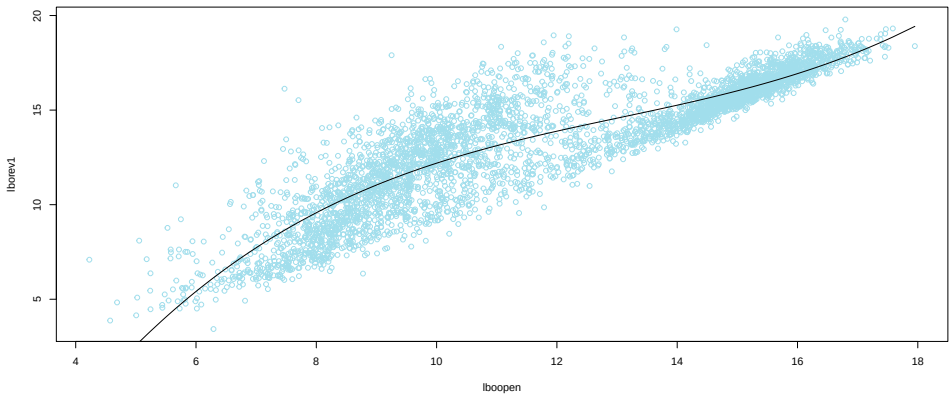
Note that for large data sets it could be more efficient (and may be essential) to calculate the polynomial terms in advance prior to using the `gamlss()` function, e.g.

`x2<-x^2; x3<-x^3`

and then use them within the `gamlss()` function, since the evaluation is then done only once.

The fitted model is displayed in Figure 2.3. The polynomial line did not fit well in the lower part of the explanatory variable `lboopen`. This behaviour, that is, the fit to be erratic in the lower or the upper end of the explanatory variable, is very common in polynomial fitting curves.

```
plot(lborev1~lboopen,col = hcl(210), data=film90)
lines(fitted(m0)[order(film90$lboopen)]~film90$lboopen[order(film90$lboopen)])
```



R code on page 53

Figure 2.3: A plot of the `film90` data together with the fitted polynomial model for the mean

f

The fitted model is given by $Y \sim NO(\hat{\mu}, \hat{\sigma})$ where $\hat{\mu} = \hat{\beta}_{01} + \hat{\beta}_{11}x + \hat{\beta}_{21}x^2 + \hat{\beta}_{31}x^3$, i.e.

$$\hat{\mu} = -22.437 + 7.174x - 0.499x^2 + 0.013x^3$$

and

$$\log(\hat{\sigma}) = \hat{\beta}_{02} = 0.3818$$

so $\hat{\sigma} = \exp(0.3818) = 1.465$ (since σ has a default log link function), where $Y =$ `lborev1` and $x =$ `lboopen`.

The `summary` function (used after convergence of the `gamlss()` function) has two ways of producing standard errors i) "vcov" and ii) "qr". The default value is `type="vcov"`. This uses the

`vcov` method for `gamlss` objects which (starting from the fitted beta parameters values given by the `gamlss()` function) defines the likelihood function (using `gen.likelihood()`) and uses this to obtain the full Hessian matrix of all the beta parameters in the model (from all the distribution parameters), i.e. β_{01}, β_{11}, β_{21}, β_{31} and β_{02} in the above model. Standard errors are obtained from the observed information matrix (the inverse of the Hessian). The standard errors obtained this way are more reliable, since they take into account the information about the interrelationship between the distribution parameters, i.e. μ and σ in the above case. On occasions, when the above procedure fails, the standard errors are obtained from `type= "qr"`, which uses the individual fits of the distribution parameters (used in the `gamlss()` algorithms) and therefore should be used with caution. The `summary()` output gives a warning when this happens. The standard errors produced this way do not take into the account the correlation between the estimates of the distribution parameters μ, σ, ν and τ, [although in the example above the estimates of the distribution parameters μ and σ of the normal distribution are asymptotically uncorrelated]

Robust (*sandwich* or "*Huber sandwich*") standard errors can be obtained using the argument `robust=TRUE` of the `summary()` function. Robust standard errors introduced by Huber [1967] and White [1980], are, in general, more reliable than the usual standard errors when the variance model is suspected not to be correct (assuming the mean model is correct). The sandwich standard errors are usually (but not always) bigger that the usual ones. Next we demonstrate how the function `vcov()` can be used to obtain the variance-covariance matrix, the correlation matrix and the (usual and robust) standard errors of the estimated parameters:

```
# the variance-covariance
print(vcov(m00), digit=3)
```

```
##              (Intercept)   lboopen I(lboopen^2) I(lboopen^3) (Intercept)
## (Intercept)     1.63e+00 -4.49e-01     3.95e-02    -1.12e-03   -2.32e-11
## lboopen        -4.49e-01  1.25e-01    -1.11e-02     3.18e-04    6.38e-12
## I(lboopen^2)    3.95e-02 -1.11e-02     1.01e-03    -2.90e-05   -5.61e-13
## I(lboopen^3)   -1.12e-03  3.18e-04    -2.90e-05     8.46e-07    1.59e-14
## (Intercept)    -2.32e-11  6.38e-12    -5.61e-13     1.59e-14    1.24e-04
```

```
# the correlation matrix
print(vcov(m00, type="cor"), digit=3)
```

```
##              (Intercept)   lboopen I(lboopen^2) I(lboopen^3) (Intercept)
## (Intercept)     1.00e+00 -9.93e-01     9.74e-01    -9.49e-01   -1.63e-09
## lboopen        -9.93e-01  1.00e+00    -9.94e-01     9.79e-01    1.62e-09
## I(lboopen^2)    9.74e-01 -9.94e-01     1.00e+00    -9.95e-01   -1.59e-09
## I(lboopen^3)   -9.49e-01  9.79e-01    -9.95e-01     1.00e+00    1.55e-09
## (Intercept)    -1.63e-09  1.62e-09    -1.59e-09     1.55e-09    1.00e+00
```

```
# standard errors
print(vcov(m00, type="se"), digits=2)
```

```
##  (Intercept)      lboopen I(lboopen^2) I(lboopen^3)  (Intercept)
##      1.27840      0.35369      0.03171      0.00092      0.01114
```

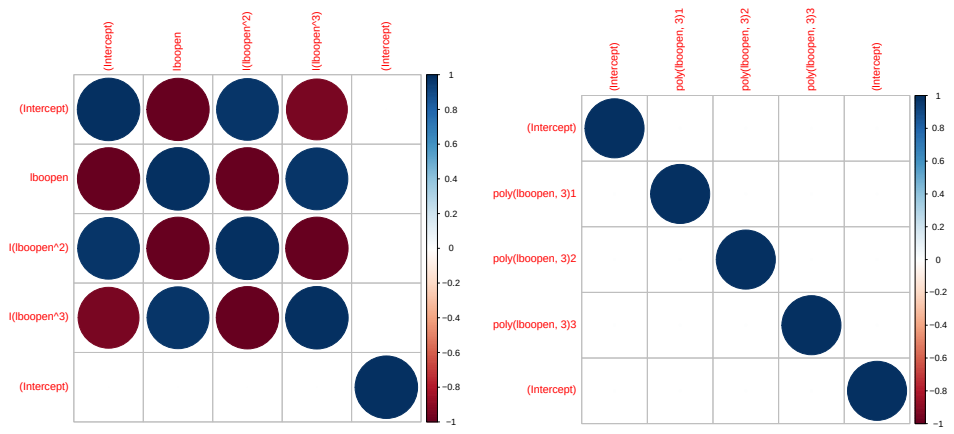
```
print(vcov(m00, type="se", robust=TRUE), digits=2)
```

```
##  (Intercept)      lboopen I(lboopen^2) I(lboopen^3)  (Intercept)
##       2.0171       0.5336       0.0455       0.0013       0.0135
```

Note that in the final row and/or column of the above output 'intercept' refers to $\hat{\beta}_{02}$ intercept of the predictor model for σ, while the first row and/or column 'intercept' refers to $\hat{\beta}_{01}$ the intercept of the predictor for μ.

Visual representation of the correlation coefficients can be obtain using the package `corrplot` and it is of some interest to compare the two fitted models with and without the `poly()` function.

```
library(corrplot)
op<-par(mfrow=c(1,2))
corrplot(vcov(m00, type="cor"))
corrplot(vcov(m0, type="cor"))
par(op)
```



R code on page 55

Figure 2.4: A plot of the correlation coefficient matrices for models `m00` on the left and `m0` on the right

Figure 2.4 shows the resulting plot. Because the μ and σ parameters in the normal distribution are information independent (i.e. asymptotically uncorrelated) the first four estimated parameters, of the model for μ are effectively not correlated with the fifth, the constant for σ, in both model `m0` and `m00`. In addition all the parameters of the μ model for `m0` are actually uncorrelated because we used orthogonal polynomials (for a model with normal errors and constant variance), but for `m00` they are highly correlated.

2.3.2 Fitting a non-parametric smoothing model

P-splines

Model `m0` is a linear parametric GAMLSS model. In order to fit μ the mean of `lborevl` with a semi-parametric model in `lboopen` using a non-parametric smoothing P-spline, Eilers and Marx [1996], use:

```
m1<-gamlss(lborev1~pb(lboopen), data=film90, family=NO)

## GAMLSS-RS iteration 1: Global Deviance = 14085.78
## GAMLSS-RS iteration 2: Global Deviance = 14085.78
```

In the smoothing function `pb()` the smoothing parameter (and therefore the effective degrees of freedom) are estimated automatically using the default local maximum likelihood method described in Rigby and Stasinopoulos [2013]. Within the `pb()` function there are also alternative ways of estimating the smoothing parameter, such as the local Generalised AIC (GAIC), and the Generalised Cross Validation (GCV).

The fitted model is displayed in Figure 2.5:

```
plot(lborev1~lboopen,col = hcl(210), data=film90)
lines(fitted(m1)[order(film90$lboopen)]~film90$lboopen[order(film90$lboopen)])
```

R code on page 56

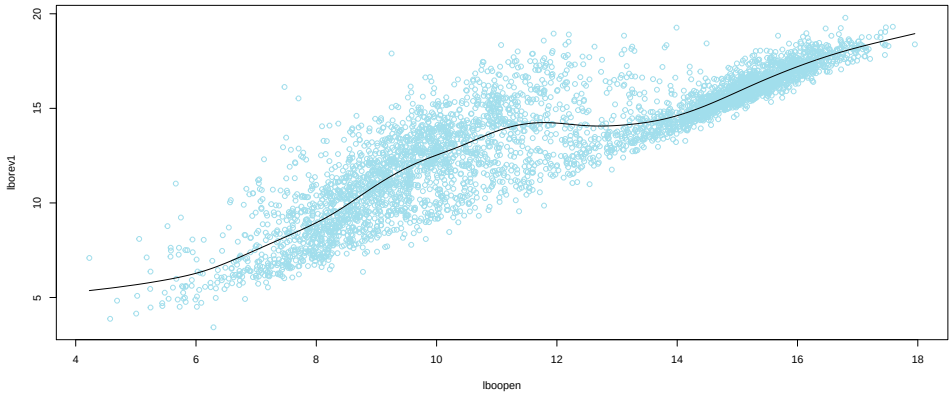


Figure 2.5: P-splines fit: a plot of the `film90` data together with the fitted smooth mean function fitted using the function `pb()`

The effective degrees of freedom fitted by the `pb()` can be obtained using the function `edf()`:

```
edf(m1, "mu")

## Effective df for mu model
## pb(lboopen)
##    12.46241

summary(m1)

## ******************************************************************
## Family:  c("NO", "Normal")
##
## Call:
## gamlss(formula = lborev1 ~ pb(lboopen), family = NO, data = film90)
##
## Fitting method: RS()
```

```
##
## ------------------------------------------------------------------
## Mu link function:  identity
## Mu Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 2.352834   0.086899   27.08   <2e-16 ***
## pb(lboopen) 0.928404   0.007137  130.08   <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)  0.32824    0.01114   29.47   <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:  4031
## Degrees of Freedom for the fit:  13.46241
##       Residual Deg. of Freedom:  4017.538
##                       at cycle:  2
##
## Global Deviance:     14085.78
##             AIC:     14112.7
##             SBC:     14197.54
## ******************************************************************
```

One of the important things to remember when fitting smooth non-parametric terms in `gamlss()` is the fact that the resulting coefficients of the smoothing term and their standard errors refer only to the linear term. For example the coefficient 0.93 and its s.e. 0.007137 in the above output should be interpreted with care. They are an artefact of the way the fitting algorithm works with the `pb()` function. It is because the linear part of the smoothing is fitted separately together with all other linear terms (in the above case with only the constant). One should try to interpret the whole smoothing function which can be obtained using `term.plot()`. Significance of smoothing terms can be obtained using the function `drop1()` but maybe be slow for large data set with a lot of fitted smoothing terms.

Important: Do not try to interpret the linear coefficients or the standard errors of the smoothing terms.

Note also that when smoothing additive terms are involved in the fitting, both methods (default and robust), used in `summary` to obtained standard errors, are questionable. The reason is because the way the function `vcov` is implemented effectively assumes that the estimated smooth-

ing terms were fixed at their estimated values. The functions `prof.dev()` and `prof.term()` can be used for obtaining more reliable individual parameter confidence intervals, by fixing the smoothing degrees of freedom at their previously selected values.

Cubic Splines

Other smoothers are also available. In order to fit a non-parametric smoothing cubic spline with 10 effective degrees of freedom on top of the constant and linear terms use

```
m2<-gamlss(lborev1~cs(lboopen,df=10), data=film90, family=NO)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 14087.15
## . . .
## GAMLSS-RS iteration 2: Global Deviance = 14087.15
```

The effective degrees of freedom used in the fitting of the `mu` parameter in the above model are 12 (one for the constant, one for the linear and 10 for smoothing). Note that the `gamlss()` notation is different to the `gam()` notation in `S-PLUS` where the equivalent model is fitted using `s(x,11)`.

The total degrees of freedom used for the above model `m2` is thirteen, i.e. 12 for `mu` the mean, and 1 for the constant scale parameter `sigma` the standard deviation of the fitted normal distribution model. The fitted values of model `m2` together with the fitted values of `m1` are displayed in Figure 2.6:

```
plot(lborev1~lboopen,col = hcl(210), data=film90)
lines(fitted(m1)[order(film90$lboopen)]~film90$lboopen[order(film90$lboopen)])
lines(fitted(m2)[order(film90$lboopen)]~film90$lboopen[order(film90$lboopen)],
col="red", lty=2,  lwd=2)
```

R code on page 58

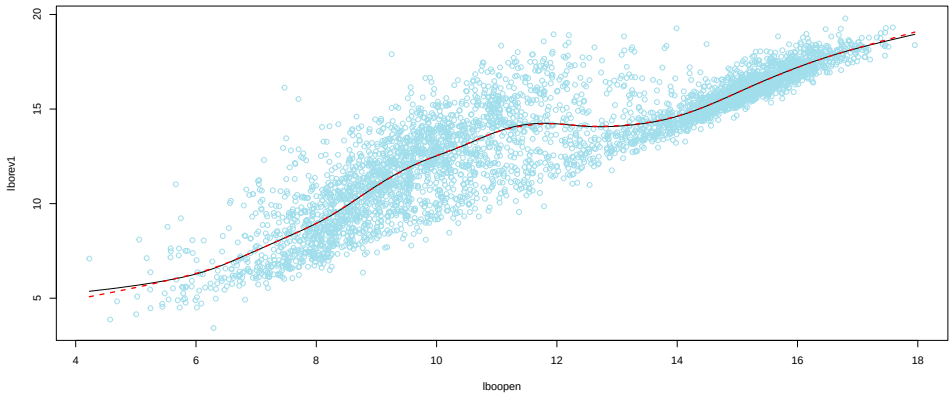


Figure 2.6: Cubic splines fit: a plot of the `film90` data together with the fitted smooth mean functions of model `m1` fitted by `pb()` (black continuous line) and model `m2` fitted by `cs()` (red dashed line).

Neural Networks

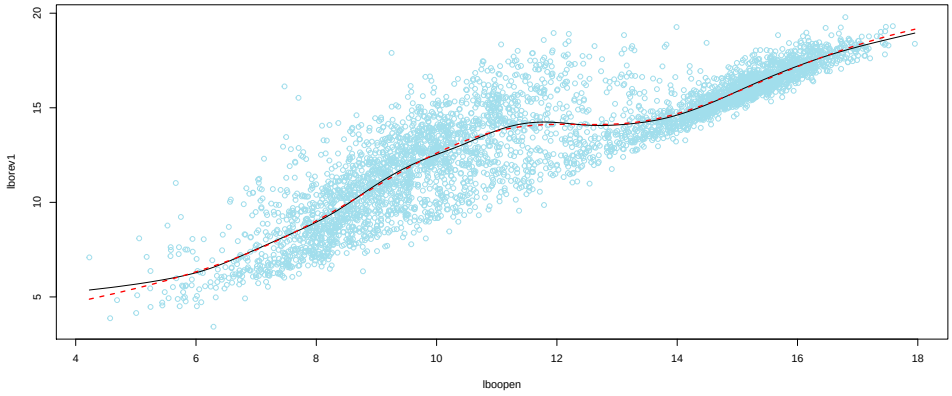
Neural networks can be considered as another type of smoother. Here a neural network smoother is fitted using an interface of the **gamlss** package with the **nnet** of Brian Ripley. The additive function to be used with `gamlss()` is called `nn()` and it is part of the package **gamlss.add** which has to be download. Here is how it works.

```
library(gamlss.add)
mnt <- gamlss(lborev1~nn(~lboopen,size=20, decay=0.1), data=film90, family=NO)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 14166.07
## . . .
## GAMLSS-RS iteration 2: Global Deviance = 14108.85
```

The fitted values of model `mnt` together with the fitted values of `m1` are displayed in Figure 2.7:

```
plot(lborev1~lboopen,col = hcl(210), data=film90)
lines(fitted(m1)[order(film90$lboopen)]~film90$lboopen[order(film90$lboopen)])
lines(fitted(mnt)[order(film90$lboopen)]~film90$lboopen[order(film90$lboopen)],
col="red", lty=2,  lwd=2)
```



R code on page 59

Figure 2.7: Neural network fit: a plot of the `film90` data together with the fitted smooth mean functions of model `m1` fitted by `pb()` (black continuous line) and the neural network model `mnt` fitted by `nn()` (red dashed line).

To get more information about the fitted neural network model use the function `getSmo()`. This function retrieves the last fitted object within the backfitting GAMLSS algorithm (in this case a `"nnet"` object). Reserved methods for the object, like `print()`, `summary()` or `coef()`, can be used to get information for the objects. Here we retrieve its 61 coefficients.

```
coef(getSmo(mnt))
```

```
##         b->h1         i1->h1         b->h2         i1->h2         b->h3
##   0.514977039  -0.122071942   0.499045016  -0.120679890  -0.528418846
```

```
## . . .
```

2.3.3 Extracting the fitted values for σ

Fitted values of the parameters of the object can be obtained using the `fitted()` function. For example `plot(lboopen, fitted(m1,"mu"))` will plot the fitted values of `mu` against x (=`lboopen`). The constant estimated scale parameter (the standard deviation of the normal distribution in this case) can be obtained:

```
fitted(m1,"sigma")[1]
```

```
##        1
## 1.388527
```

where `[1]` indicates the first value of the vector. The same values can be obtained using the more general function `predict()`:

```
predict(m1,what="sigma", type="response")[1]
```

```
##        1
## 1.388527
```

The function `predict()` can also be used to predict the response variable distribution parameters for both old and new data values of the explanatory variables.

2.3.4 Modelling both μ and σ

To model the predictors of both the mean, μ, and the scale parameter, σ (i.e. μ and $\log\sigma$), as non-parametric smoothing cubic spline functions of `x` (with a normal distribution for the response Y) use:

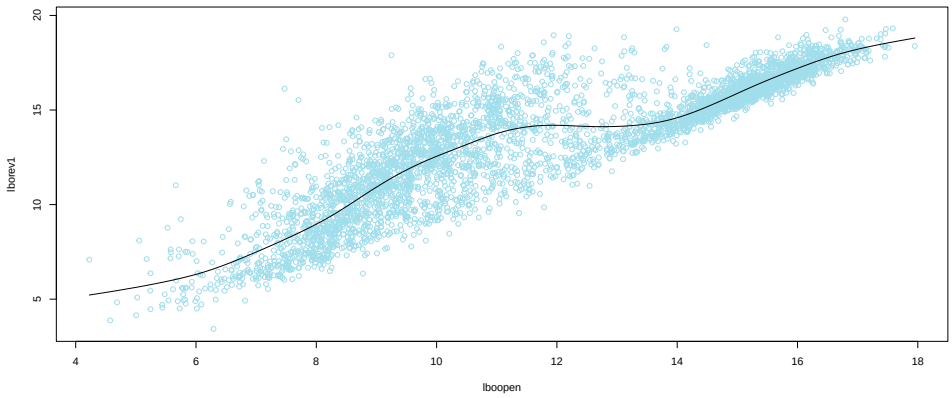
```
m3 <- gamlss(lborev1~pb(lboopen),sigma.formula=~pb(lboopen),
data=film90, family=NO)
edfAll(m3)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 12224
## . . .
## GAMLSS-RS iteration 4: Global Deviance = 12227
## $mu
## pb(lboopen)
##       12.41
##
## $sigma
## pb(lboopen)
##       10.96
```

This time we used the function `edfAll()` to obtain the effective degrees of freedom for all parameters. The estimated total degrees of freedom for smoothing are 12.41 and 10.96 for μ and σ respectively.

The fitted model for μ, the mean of the response variable `lborev1`, is displayed in Figure 2.8:

```
plot(lborev1~lboopen,col = hcl(210), data=film90)
lines(fitted(m3)[order(film90$lboopen)]~film90$lboopen[order(film90$lboopen)])
```



R code on page 61

Figure 2.8: Fitted mean and variance model: a plot of the `film90` data together with the fitted smooth mean function of the model `m3` where both the mean and variance models are fitted using `pb()`.

loess

If you wish to use loess curves, see Cleveland and Devlin [1988], instead of cubic or penalised splines use:

```
m4 <- gamlss(lborev1~lo(~lboopen,span=.4), sigma.formula=~lo(~lboopen,span=.4),
data=film90, family=NO)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 12250
## . . .
## GAMLSS-RS iteration 4: Global Deviance = 12250
```

2.3.5 Diagnostic plots

The function `resid(abd2)` (an abbreviation of `residuals()`) can be used to obtain the fitted (normalized randomized quantile) residuals of a model, subsequently just called residuals throughout this introduction. The residuals only need to be randomized for discrete distributions, see Dunn and Smyth [1996] and the Chapter 12 for more details. Residuals plots can be obtained using `plot()`.

```
plot(m2)
```

```
## ******************************************************************
##        Summary of the Quantile Residuals
##                             mean   =  4.585806e-06
```

```
##                          variance   =  1.000248
##                 coef. of skewness   =  0.3662528
##                 coef. of kurtosis   =  4.376454
## Filliben correlation coefficient   =  0.9877495
## ******************************************************************
```

R code on page 62

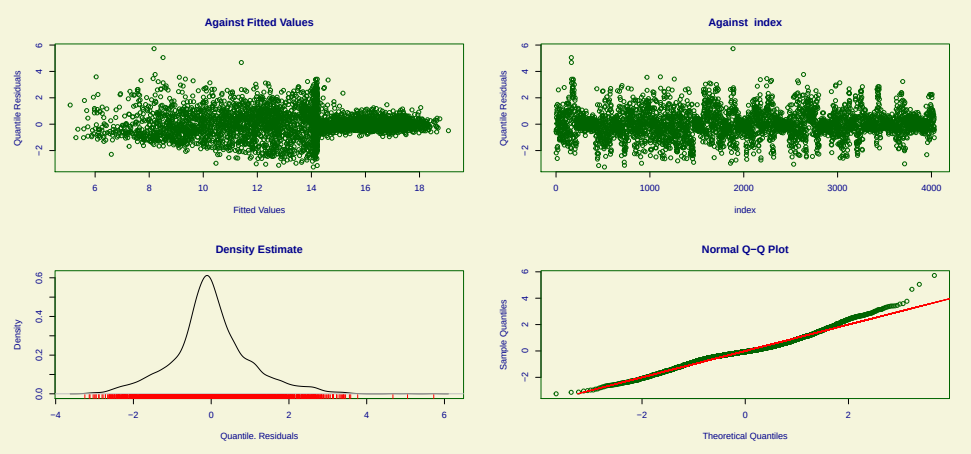


Figure 2.9: Residual plot from the fitted normal model `m2` with model `pb(x)` for both μ and $\log \sigma$.

See Figure 2.9 for the plot. Figure 2.9 shows plots of the (normalized quantile) residuals: i) against the fitted values ii) against a index iii) a non-parametric kernel density estimate iv) a normal Q-Q plot.

Note that the `plot()` function does not produce additive term plots [as it does for example in the `gam()` function of the package **mgcv**] in **R**. The function which does this in the **gamlss** package is `term.plot()`.

A worm plot of the residuals, see van Buuren and Fredriks [2001], can be obtained by using the `wp()` function:

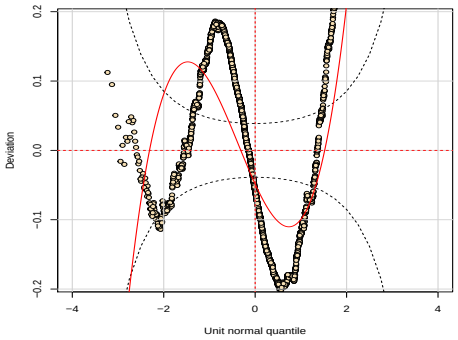
```
wp(m2)
```

```
## Warning in wp(m2):  Some points are missed out
## increase the y limits using ylim.all
```

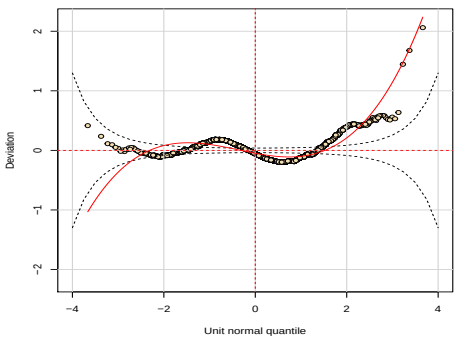
See Figure 2.10(a) for the plot. To include all points in the worm plot change the 'Deviation' axis range by increasing the value of `ylim.all`:

```
  wp(m2, ylim.all=2.2)
```

Since there is no warning message, all points have been included in the worm plot. See Figure 2.10(b) for the plot. The default worm plot above is a de-trended normal Q-Q plot of the residuals, and indicates a inadequacy in modelling the distribution, since many points plotted lie outside the (dotted) pointwise 95% confidence bands.



(a)



(b)

R code on page **??**

Figure 2.10: Worm plot from model `m2`.

2.3.6 Fitting different distributions

If you wish to use a different distribution instead of the normal, use the option `family` of the function `gamlss()`. For example to fit the Box-Cox-Cole-Green (BCCG) a 3-parameter distribution use:

```
m5 <-gamlss(lborev1~pb(lboopen), sigma.formula=~pb(lboopen),
            nu.formula=~pb(lboopen),
            data=film90, family=BCCG)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 11845
## . . .
## GAMLSS-RS iteration 17: Global Deviance = 11769
```

To fit the Box-Cox Power Exponential (BCPE) a 4-parameter distribution) try:

```
m6 <-gamlss(lborev1~pb(lboopen), sigma.formula=~pb(lboopen),
            nu.formula=~pb(lboopen), tau.formula=~pb(lboopen),
            data=film90, start.from=m5, family=BCPE)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 11699
## . . .
## GAMLSS-RS iteration 19: Global Deviance = 11696
```

Note that we have used the `gamlss()` argument `start.from=m5` to start the iterations from the previous fitted `m5` model. The details of all the distributions currently available in `gamlss()` are given in the book "Distribution for Location scale and shape".

2.3.7 Selection between models

Different models can be compared using their global deviances, $GD = -2\hat{\ell}$, (if they are nested) or using a generalised Akaike information criterion, $GAIC = -2\hat{\ell} + (k.df)$, where $\hat{\ell} = \sum_{i=1}^{n} \log f(y_i|\hat{\mu}_i, \hat{\sigma}_i, \hat{\nu}_i, \hat{\tau}_i)$ is the fitted log-likelihood function and k is a required penalty,

e.g. $k = 2$ for the usual Akaike information criterion or $k = log(n)$ for the Schwartz Bayesian criterion or $k = 3.84$ (corresponding to a Chi-squared test with one degree of freedom for a single parameter, since $\chi^2_{1,0.05} = 3.84$). The function `deviance()` provides the global deviance of the model. Note that the GAMLSS global deviance is different from the deviance that is provided by the functions `glm()` and `gam()` in **R**. The global deviance is **exactly** minus twice the fitted log likelihood function, *including* all constant terms in the log-likelihood. The `glm()` deviance is calculated as a deviation from the saturated model and it does not include 'constant' terms (which do not depend on the mean of distribution but depend in scale parameter) in the fitted log likelihood and so cannot be used to compare different distributions. To obtain the generalised Akaike information criterion use the functions `AIC()` or `GAIC()`. The functions are identical. For example to compare the models `m0` to `m6` use:

```
AIC(m0,m1,m2,m3,m4,m5,m6)
```

```
##          df      AIC
## m6 45.82525 11787.30
## m5 36.84390 11842.87
## m3 23.37377 12273.40
## m4 18.04177 12286.01
## m1 13.46241 14112.70
## m2 13.00200 14113.16
## m0  5.00000 14527.65
```

The GAIC function uses default penalty $k = 2$, giving the usual Akaike information criterion (AIC). Hence the usual AIC [equivalent to GAIC($k = 2$)] selects model `m6` as the best model (since it has the smallest value of AIC). If you wish to change the penalty in `GAIC()` use the argument `k`.

```
AIC(m0,m1,m2,m3,m4,m5,m6, k=log(4031))
```

```
##          df      AIC
## m5 36.84390 12075.05
## m6 45.82525 12076.08
## m4 18.04177 12399.71
## m3 23.37377 12420.69
## m2 13.00200 14195.09
## m1 13.46241 14197.54
## m0  5.00000 14559.15
```

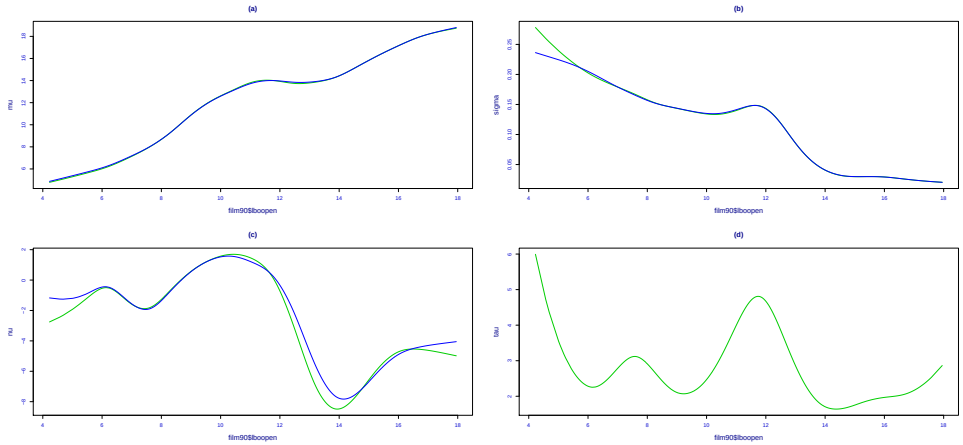
In this case with GAIC($k = log(n)$) we have the Bayesian Information Criterion (BIC). Models selected using BIC are generallly simpler that models selected using AIC. This is the case here where model `m5` is selected.

Other criteria based on training, validation and test samples are discussed on Chapter 11.

2.3.8 Chosen Model

Using the criterion GAIC with $k = 2$ (i.e. the usual AIC criterion), model `m6` is selected with $Y =$ `lborev` $\sim BCPE(\mu, \sigma, \nu, \tau)$ where each of μ, σ, ν and τ are modelled as smooth functions of the explanatory variable $x =$ `lboopen`. The fitted smooth functions for both `m5` and `m6` models are shown in Figure 2.11.

```
fittedPlot(m5, m6, x=film90$lboopen)
```

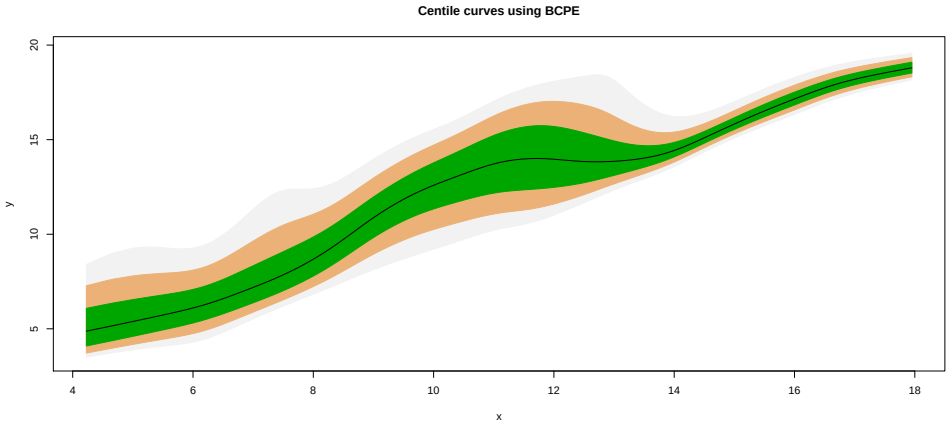


R code on page 65

Figure 2.11: A plot of the smooth fitted values for all the parameters (a) μ, (b) σ, (c) ν and (d) τ from models `m5` and `m6`.

Since, in this example, only one explanatory variable is used in the fit, centiles estimates for the fitted distribution can be shown using the functions `centiles()` or `centiles.fan()`.

```
centiles.fan(m6,  xvar=film90$lboopen,  cent=c(3,10,25,50,75,90,97),
colors="terrain")
```



R code on page 65

Figure 2.12: A centile fan plot for the fitted `m6` model showing the 3, 10, 25, 50, 75, 90 and 97 centiles for the fitted BCPE distribution.

The next plot is also showing how the fitted conditional distribution for the response variable `lborev1` changes according to variable `lboopen`. The function `plotSimpleGamlss()` from the package **gamlss.util** is used here.

```
library(gamlss.util)
library(colorspace)
plotSimpleGamlss(lborev1,lboopen, model=m6,   data=film90, x.val=seq(6,16,2),
val=5, N=1000, ylim=c(0,25), cols=heat_hcl(100))

## new prediction
## new prediction

## Warning in predict.gamlss(object, newdata = newdata, what = "nu", type = type,
:  There is a discrepancy  between the original and the re-fit
##  used to achieve 'safe' predictions
##

## new prediction
## new prediction
```

R code on page 66

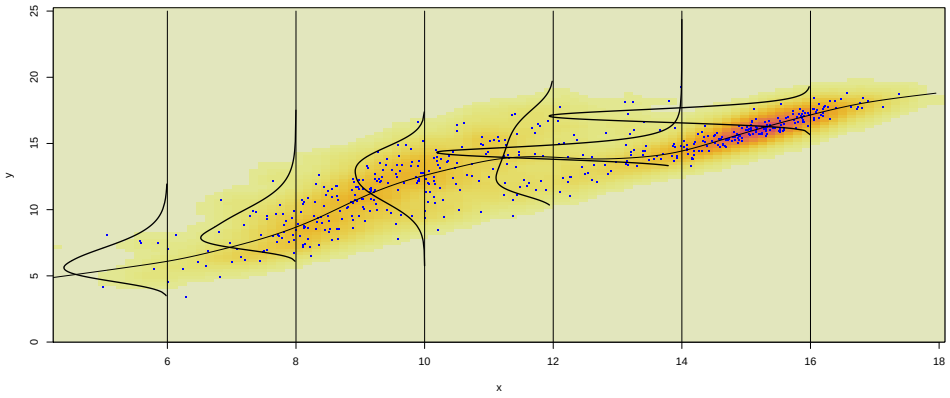


Figure 2.13: A plot showing how the fitted conditional distribution of the response variable `lborev1` changes for different values of the explanatory variable `lboopen`.

The resulting plot is shown in Figure 2.13.This plot highlighted how the fitted conditional distribution of `lborev1` changes with `lboopen`. That is the essence of the GAMLSS modelling.

Important: Within GAMLSS the shape of conditional distribution of the response variable can vary according to the values of the explanatory variables.

Part II

The R implementation: algorithms and functions

Chapter 3

The Algorithms

This chapter:

- redefines the GAMLSS models and
- describes the two algorithms for maximising the penalised log-likelihood function.

The material provided here will help the user to get an inside view of how the fitting algorithms of GAMLSS are working.

3.1 Introduction

The GAMLSS model was first introduced in Section 1.7 of Chapter 1 as

$$\begin{aligned} \mathbf{y} &\overset{\text{ind}}{\sim} D(\boldsymbol{\mu},\boldsymbol{\sigma},\boldsymbol{\nu},\boldsymbol{\tau}) \\ g_1(\boldsymbol{\mu}) &= \mathbf{X}_1\boldsymbol{\beta}_1 + s_{11}(\mathbf{x}_{11}) + \ldots + s_{1J_1}(\mathbf{x}_{1J_1}) \\ g_2(\boldsymbol{\sigma}) &= \mathbf{X}_2\boldsymbol{\beta}_2 + s_{21}(\mathbf{x}_{21}) + \ldots + s_{2J_2}(\mathbf{x}_{2J_2}) \\ g_3(\boldsymbol{\nu}) &= \mathbf{X}_3\boldsymbol{\beta}_3 + s_{31}(\mathbf{x}_{31}) + \ldots + s_{3J_3}(\mathbf{x}_{3J_3}) \\ g_4(\boldsymbol{\tau}) &= \mathbf{X}_4\boldsymbol{\beta}_4 + s_{41}(\mathbf{x}_{41}) + \ldots + s_{4J_4}(\mathbf{x}_{4J_4}) \end{aligned} \tag{3.1}$$

where $D(\boldsymbol{\mu},\boldsymbol{\sigma},\boldsymbol{\nu},\boldsymbol{\tau})$ is the distribution of the response variable $\mathbf{y}$, $\mathbf{X}_k$ for $k = 1,2,3,4$ are the design matrices incorporating the linear additive terms in the model (see Chapter 8), $\boldsymbol{\beta}_k$ for $k = 1,2,3,4$ are the linear parameters and $s_{kj}(\mathbf{x}_{kj})$ represent different smoothing functions for different explanatory variables $\mathbf{x}_{kj}$ for $k = 1,2,3,4$ and $j = 1,\ldots,J_k$.

It turns out that most of a smooth functions used within GAMLSS can be written in the form of $s(\mathbf{x}) = \mathbf{Z}\boldsymbol{\gamma}$ where $\mathbf{Z}$ is the basis matrix which depends on the explanatory variable $\mathbf{x}$, (see Chapter **??**). The $\boldsymbol{\gamma}$ is a parameter vector to be estimated, subject to a quadratic a penalty of the form $\lambda\boldsymbol{\gamma}^\top\mathbf{G}\boldsymbol{\gamma}$, for a known matrix $\mathbf{G} = \mathbf{D}^\top\mathbf{D}$ and where the hyper-parameter λ regulates the amount of smoothing needed for the fit. We shall refer to functions in this form as *penalised smooth* functions (or *penalised smoothers*). Penalised smoothers are the subject of Chapter **??** where it is shown that different formulations for the $\mathbf{Z}$'s and for the $\mathbf{D}$'s lead to different types

of smoothing functions with different statistical properties. We can now rewrite the GAMLSS model in equation (3.1) as:

$$\begin{aligned}
\mathbf{y} &\overset{\text{ind}}{\sim} D(\boldsymbol{\mu},\boldsymbol{\sigma},\boldsymbol{\nu},\boldsymbol{\tau}) \\
g_1(\boldsymbol{\mu}) &= \mathbf{X}_1\boldsymbol{\beta}_1 + \mathbf{Z}_{11}\boldsymbol{\gamma}_{11} + \ldots + \mathbf{Z}_{1k_1}\boldsymbol{\gamma}_{1J_1} \\
g_2(\boldsymbol{\sigma}) &= \mathbf{X}_2\boldsymbol{\beta}_2 + \mathbf{Z}_{21}\boldsymbol{\gamma}_{21} + \ldots + \mathbf{Z}_{2k_2}\boldsymbol{\gamma}_{2J_2} \\
g_3(\boldsymbol{\nu}) &= \mathbf{X}_3\boldsymbol{\beta}_3 + \mathbf{Z}_{31}\boldsymbol{\gamma}_{31} + \ldots + \mathbf{Z}_{3k_3}\boldsymbol{\gamma}_{3J_3} \\
g_4(\boldsymbol{\tau}) &= \mathbf{X}_4\boldsymbol{\beta}_4 + \mathbf{Z}_{41}\boldsymbol{\gamma}_{41} + \ldots + \mathbf{Z}_{4k_4}\boldsymbol{\gamma}_{4J_4}, \qquad (3.2)
\end{aligned}$$

subject to the penalty

$$\sum_{k=1}^{4}\sum_{j=1}^{J_k}\lambda_{kj}\boldsymbol{\gamma}_{kj}^{\top}\mathbf{G}_{kj}\boldsymbol{\gamma}_{kj}. \qquad (3.3)$$

If there are no smooth functions in the model, the model is simplified to:

$$\begin{aligned}
\mathbf{y} &\overset{\text{ind}}{\sim} D(\boldsymbol{\mu},\boldsymbol{\sigma},\boldsymbol{\nu},\boldsymbol{\tau}) \\
g_1(\boldsymbol{\mu}) &= \mathbf{X}_1\boldsymbol{\beta}_1 \\
g_2(\boldsymbol{\sigma}) &= \mathbf{X}_2\boldsymbol{\beta}_2 \\
g_3(\boldsymbol{\nu}) &= \mathbf{X}_3\boldsymbol{\beta}_3 \\
g_4(\boldsymbol{\tau}) &= \mathbf{X}_4\boldsymbol{\beta}_4. \qquad (3.4)
\end{aligned}$$

We refer to model (3.4) as the *parametric* GAMLSS models while the model defined by equations (3.2) and (3.3) as the *non-parametric* GAMLSS model. Within the `R` implementation, the parametric GAMLSS model (3.4) is fitted by maximum likelihood estimation, while the more general non-parametric model of (3.2) and (3.3) is fitted by maximum penalised likelihood estimation. The log likelihood function for the GAMLSS model (3.4) under the assumption that observations in the response variables are independent is given by

$$\ell = \sum_{i=1}^{n}\log f(y_i|\mu_i,\sigma_i,\nu_i,\tau_i) \qquad (3.5)$$

where $f()$ represent the probability (density) function of the response variable. The penalised log-likelihood function for models (3.2) and (3.3) is given by

$$\ell_p = \ell - \frac{1}{2}\sum_{k=1}^{4}\sum_{j=1}^{J_k}\lambda_{kj}\boldsymbol{\gamma}_{kj}^{\top}\mathbf{G}_{kj}\boldsymbol{\gamma}_{kj} \qquad (3.6)$$

Note we will need estimates for the 'betas',

$$\boldsymbol{\beta} = (\boldsymbol{\beta}_1,\boldsymbol{\beta}_2,\boldsymbol{\beta}_3,\boldsymbol{\beta}_4)$$

the 'gammas',

$$\boldsymbol{\gamma} = (\boldsymbol{\gamma}_{11},\ldots,\boldsymbol{\gamma}_{1J_1},\boldsymbol{\gamma}_{21},\ldots,\boldsymbol{\gamma}_{4J_4}),$$

and the 'lambdas'

$$\boldsymbol{\lambda} = (\boldsymbol{\lambda}_{11},\ldots,\boldsymbol{\lambda}_{1J_1},\boldsymbol{\lambda}_{21},\ldots,\boldsymbol{\lambda}_{4J_4}).$$

There are two basic algorithms for fitting the parametric model (3.4) or the non-parametric model GAMLSS model of equations (3.2) and (3.3), the `RS` and the `CG` algorithms. The two algorithms will be explained in the next section.

3.2 Estimating β and γ for fixed λ

Rigby and Stasinopoulos [2005] provided two basic algorithms for maximising the penalised log likelihood given in (3.6) with respect to $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ for a given $\boldsymbol{\lambda}$:

- The `CG` algorithm which is a generalisation of the Cole and Green [1992] algorithm. This algorithm requires information about the first and (expected or approximated) second and cross derivatives of the log-likelihood function with respect to the distribution parameters $\boldsymbol{\theta} = (\mu, \sigma, \nu, \tau)$ for a four parameter distribution.
- The `RS` algorithm which is a generalisation of the the algorithm used by Rigby and Stasinopoulos [1996a,b] for fitting a mean and dispersion additive models, (MADAM). This algorithm does not use the cross derivatives of the log-likelihood.

Appendix C of Rigby and Stasinopoulos [2005] shows that both algorithms lead, for given $\boldsymbol{\lambda}$ hyper-parameters, to the maximum penalised log likelihood estimates for the betas and the gammas, i.e. $\hat{\boldsymbol{\beta}}$ and $\hat{\boldsymbol{\gamma}}$.

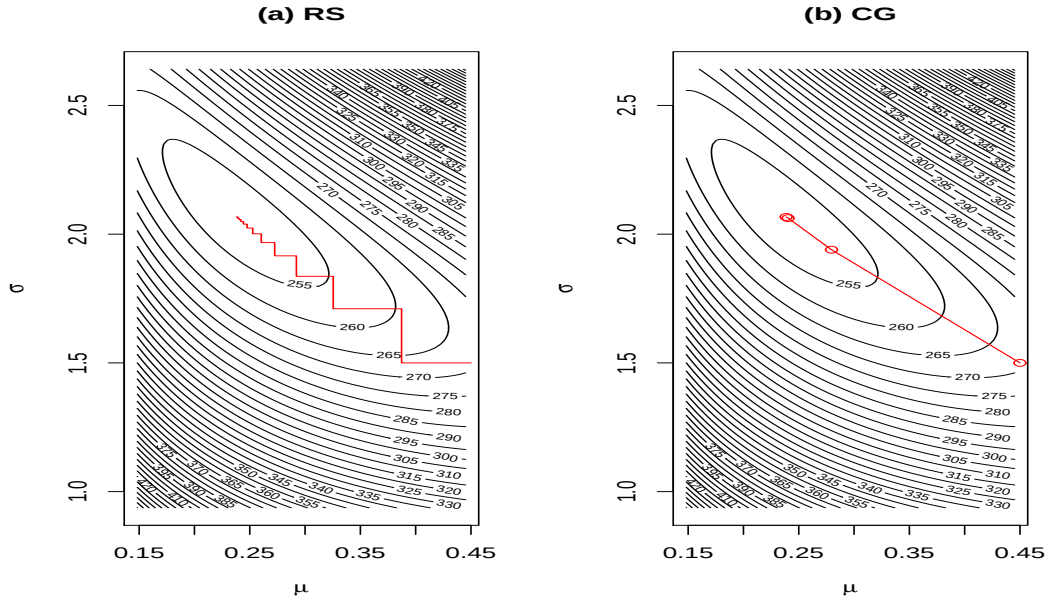


Figure 3.1: Showing how the two GAMLSS algorithms (a) RS and (b) CG reach the maximum.

Figure 3.1 demonstrates the different ways in which the two algorithms reach the maximum log likelihood parameter estimates. The contours are equal global deviance (GD) contours (equal to minus twice the log likelihood). Hence maximising the log likelihood is equivalent to minimising the global deviance. The two figures are generated using a random sample from a Weibull, `WEI`(μ, σ), distribution. The `RS` algorithm maximizes the (penalized) likelihood over each of μ, σ, ν and τ in turn, cycling until convergence. For example in Figure 3.1(a) the global deviance is minimized (and hence the likelihood is maximized) over each of μ and σ in turn, alternating until convergence. The `CG` algorithm has the ability, since it uses the information about the cross derivatives, to jointly update (μ, σ) as demonstrated in Figure 3.1(b). On the

basis of the evidence in Figure 3.1 it sems that the CG algorithm should be preferable, but in practice this is not the case. The CG algorithm is rather unstable especially at the beginning of the iterations and diverges easily. The RS algorithm is generally a lot more stable and in most cases faster, so it is used as the default. Note though that for highly correlated distribution parameters the RS algorithm can be slower and may converge early before reaching the maximum log likelihood.

The RS and CG algorithms are implemented in the option `method` in the function `gamlss()` where a combination of both algorithms is also allowed using the `mixed()` function, see 4.2.1. The `mixed()` function uses the RS algorithm for the early iterations but later switches to the CG algorithm. This is recommended for highly correlated distribution parameters.

Next we describe the two algorithms in more detail.

3.2.1 The RS algorithm

The RS algorithm can be described using the following three nested components:

- the *outer iteration*, described in Figure 3.2, which calls
- the *inner iteration* (or local scoring or GLIM algorithm), described in Figure 3.3, which calls
- the *modified backfitting* algorithm, described in Figure 3.4.

The outer iteration calls repeatedly the inner iteration, which in turn calls repeatedly the modified backfitting algorithm. Convergence occurs when all three algorithms have converged.

The outer iteration (called the GAMLSS iteration)

Figure 3.2 describes the outer iteration diagrammatically. After some initialization for the parameter vectors of length n say $\boldsymbol{\mu}_0$, $\boldsymbol{\sigma}_0$, $\boldsymbol{\nu}_0$ and $\boldsymbol{\tau}_0$ for $\boldsymbol{\mu}$, $\boldsymbol{\sigma}$, $\boldsymbol{\nu}$ and $\boldsymbol{\tau}$, the outer iteration proceeds as follows:

1. fit a model for $\boldsymbol{\mu}$ [i.e. maximise the (penalised) log likelihood over $\boldsymbol{\mu}$] given the latest estimates $\hat{\boldsymbol{\sigma}}$, $\hat{\boldsymbol{\nu}}$ and $\hat{\boldsymbol{\tau}}$, then
2. fit a model for $\boldsymbol{\sigma}$ given the latest estimates $\hat{\boldsymbol{\mu}}$, $\hat{\boldsymbol{\nu}}$ and $\hat{\boldsymbol{\tau}}$, then
3. fit a model for $\boldsymbol{\nu}$ given the latest estimates $\hat{\boldsymbol{\mu}}$, $\hat{\boldsymbol{\sigma}}$ and $\hat{\boldsymbol{\tau}}$, and finally
4. fit a model for $\boldsymbol{\tau}$ given the latest estimates $\hat{\boldsymbol{\mu}}$, $\hat{\boldsymbol{\sigma}}$ and $\hat{\boldsymbol{\nu}}$.

Then it calculates the global deviance (equal to minus twice the current fitted log likelihood). If the global deviance has converged then the algorithm stops, otherwise it repeats the process.

Note that the algorithm only needs initial values for the distribution parameters, $\boldsymbol{\theta} = (\theta_1, \theta_3, \theta_3, \theta_4) = (\mu, \sigma, \nu, \tau)$ rather than for the $\boldsymbol{\beta}$ parameters. The algorithm has generally been found to be stable and fast using very simple starting values (e.g. constants) for the $\boldsymbol{\theta}$ parameters. Default values can be changed by the user if necessary (see the arguments of the `gamlss()` function).

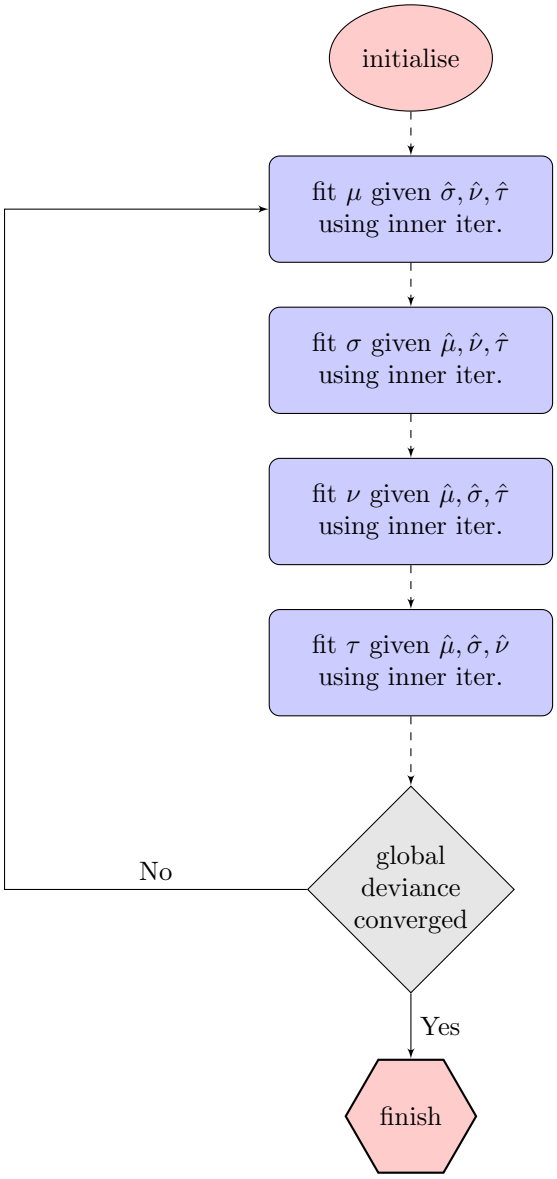


Figure 3.2: Diagram showing the outer-iteration within the GAMLSS RS algorithm

The inner iteration (called the GLM or GLIM iteration)

Now for each fitting of a distribuion parameter, θ_k for $k = 1, 2, 3, 4$, the inner iteration is used. The inner iteration is a local scoring algorithm very similar to the one used to fit generalised lineal models (GLM). This explains also the name 'GLIM algorithm'. GLIM was a computer package belonging to the Royal Statistical Society suitable of fitting GLM's. The first ever version of GAMLSS in the late 90's was written in GLIM which by now is almost an extinct species.

The idea of the local scoring algorithm is repeated weighted fits to a modified response variable using modified weights until convergence when the maximum is reached. This procedure within the GLM literature is also known as Iterative Reweighted Least Squares (IRLS).

The modified (iterative) response variable (sometimes called the *working variable*) for fitting the parameter θ_k is given by

$$\mathbf{z}_k = \boldsymbol{\eta}_k + \mathbf{w}_k^{-1} \bullet \mathbf{u}_k \tag{3.7}$$

where $\mathbf{z}_k$, $\boldsymbol{\eta}_k$, $\mathbf{w}_k$ and $\mathbf{u}_k$ are all vectors of length n, e.g. $\mathbf{w}_k = (w_{k1}, w_{k2}, \ldots, w_{kn})\top$, and $w_k^{-1} \bullet \mathbf{u}_k = (w_{k1}u_{k1}, w_{k2}u_{k2}, \ldots, w_{kn}u_{kn})^\top$ is the Hadamard element by element product, and $\boldsymbol{\eta}_k = g_k(\boldsymbol{\theta}_k)$ is the *predictor* of the k^{th} parameter θ_k for $k = 1, 2, 3, 4$, corresponding to parameters μ, σ, ν and τ respectively, and

$$\mathbf{u}_k = \frac{\partial \ell}{\partial \boldsymbol{\eta}_k} = \left(\frac{\partial \ell}{\partial \boldsymbol{\theta}_k}\right) \bullet \left(\frac{d\boldsymbol{\theta}_k}{d\boldsymbol{\eta}_k}\right)$$

is the score function (the first derivative of the log-likelihood with respect to the predictor). Note $d\boldsymbol{\theta}_k/d\boldsymbol{\eta}_k$ is a vector of length n with elements $d\boldsymbol{\theta}_{ki}/d\boldsymbol{\eta}_{ki}$ for $i = 1, \ldots, n$. The $\mathbf{w}_k$ are the *iterative weights* for $k = 1, 2, 3, 4$ defined in one of the three different ways:

$$\mathbf{w}_k = -\mathbf{f}_k \bullet \left(\frac{d\boldsymbol{\theta}_k}{d\boldsymbol{\eta}_k}\right) \bullet \left(\frac{d\boldsymbol{\theta}_k}{d\boldsymbol{\eta}_k}\right), \tag{3.8}$$

where there are three different ways to define $\mathbf{f}_k$ depending on the information available for the specific distribution:

$$\mathbf{f}_k = \begin{cases} E\left[\frac{\partial^2 \ell}{\partial \boldsymbol{\theta}_k^2}\right], \text{if the expectation exists, leading to a Fisher's scoring algorithm,} \\ \qquad where \frac{\partial^2 \ell}{\partial \boldsymbol{\theta}_k^2} \text{is a vector of length} n \text{with elements} \frac{\partial^2 \ell}{\partial \boldsymbol{\theta}_{ki}}^2 \text{for } i = 1, 2, \ldots, n \\ \frac{\partial^2 \ell}{\partial \boldsymbol{\theta}_k^2}, \text{leading to the standard Newton-Raphson scoring algorithm} \\ -\left(\frac{\partial \ell}{\partial \boldsymbol{\theta}_k}\right) \bullet \left(\frac{\partial \ell}{\partial \boldsymbol{\theta}_k}\right) \text{leading to a quasi Newton-Raphson scoring algorithm.} \end{cases}$$

Occasionally numerical derivatives are used to define $\mathbf{f}$, but this, in general, slows down the algorithm and can make it more unstable. [Note that $\frac{\partial^2 \ell}{\partial \boldsymbol{\theta}_k \partial \boldsymbol{\theta}_k^\top}$ is not used in the current implementation of the algorithm in `gamlss()` because it can give negative weights which is not allowed in the backfitting].

Figure 3.3 describes the local scoring algorithm. Given the current estimates for all the parameters $\hat{\mu}, \hat{\sigma}, \hat{\nu}$ and $\hat{\tau}$ the iterative weights and iterative working variable for the current distribution

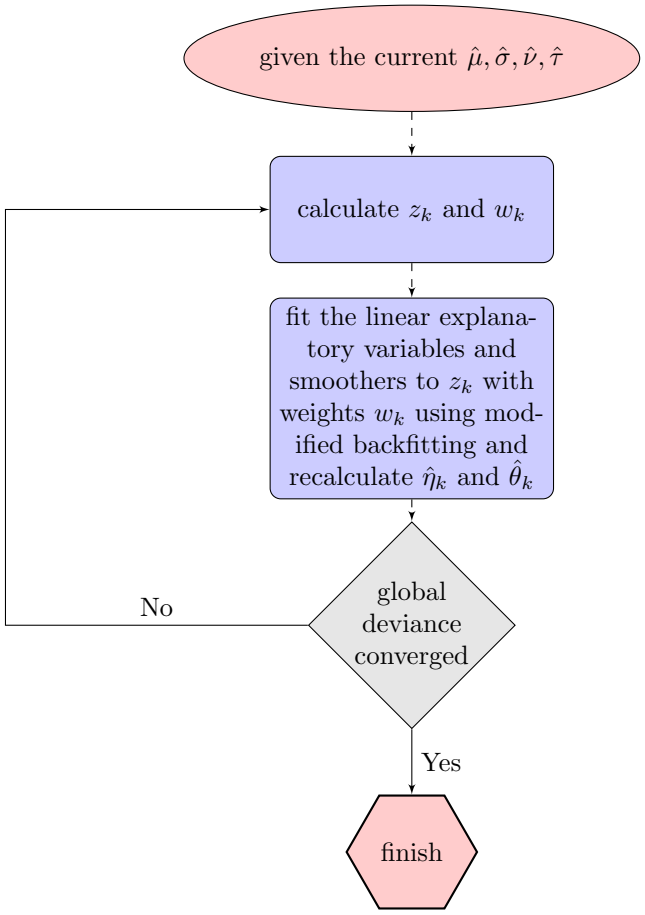


Figure 3.3: Diagram showing the inner iteration (or GLIM iteration) within the GAMLSS RS algorithm.

parameter θ are recalculated and used in a weighted fit against all the explanatory variables needed for this parameter. This is repeated until there is no change in the global deviance. (Note that other parameters are fixed at their current values throughout the inner iteration).

There are two tuning method within the inner iteration algorithm to avoid over jumping (i.e. going further away from the maximum). Both of them adjust the predictor η. The first is based on the *step* parameter $0 < \phi \leq 1$ which can be specified by an argument eg `mu.step` in the `gamlss` functions. To demonstrate how it works let η_o, η_f and η_n be the predictor from the previous iteration fit, from the current iteration fit and the proposed new predictor respectively, then $\eta_n = \phi\eta_f + (1-\phi)\eta_o$. The default value for each step parameter is 1. The second method *automatically* halvex the step (up to 5 runs) η to $\eta_n = (\eta_f + \eta_o)/2$ if the deviance increases.

The modified backfitting algorithm

The estimation of the beta and gamma parameters is done within the modified backfitting part of the algorithm. The backfitting algorithm is a version of the Gauss-Seidel algorithm Hastie and Tibshirani [1990]. (Some people can say that the whole RS algorithm is a Gauss-Seidel algorithm). The modification is that for most penalised smoothers the design matrix $\mathbf{X}$ used for the linear relationships contains the linear part of the relevant x-variable. That helps the convergence of the algorithm. The components that the backfitting algorithm needs are i) a good Weighted Least Squares (WLS) algorithm and ii) a good Weighted Penalised Least Squares (WPLS) algorithm. [In section ??? we do show that all the smoothers with a quadratic penalty can be fitted by least squares using an augmented data model.]

The backfitting algorithm works as follows. We wish to fit linear explanatory variables and smoothers to $\mathbf{z}_k$ with working weights $\mathbf{w}_k$ using backfitting (within the inner iteration for updating distribution parameter θ_k). How the process works within the `RS` algorithm is demonstrated in Figure 3.4 where $\mathbf{X}_k$ represents the design matrix for the linear part of the model with coefficients $\boldsymbol{\beta}_k$ and for simplicity we assume only two smoothers with parameter sets $\boldsymbol{\gamma}_{k1}$ and $\boldsymbol{\gamma}_{k2}$ and with basis matrices $\mathbf{Z}_{k1}$ and $\mathbf{Z}_{k2}$ respectively.

For given iterative weights $\mathbf{w}_k$ and working response variable $\mathbf{z}_k$ and previously initialised or estimated values for the coefficients of the two smoothers $\hat{\boldsymbol{\gamma}}_{k1}$ and $\hat{\boldsymbol{\gamma}}_{k2}$ calculate the partial residuals for the beta parameters $\boldsymbol{\beta}_k$ (equivalently offsetting for $\hat{\boldsymbol{\gamma}}_{k1}$ and $\hat{\boldsymbol{\gamma}}_{k2}$) and fit a WLS to the residuals to obtain a new estimate for $\hat{\boldsymbol{\beta}}_k$. Now obtain the partial residual with respect to the first smoother and use PWLS to obtain a new estimate of $\hat{\boldsymbol{\gamma}}_{k1}$. Then obtain the partial residual with respect to the second smoother and used PWLS to obtain a new estimate of $\hat{\boldsymbol{\gamma}}_{k2}$. Repeat the process until the $\hat{\boldsymbol{\beta}}_k$, $\hat{\boldsymbol{\gamma}}_{k1}$ and $\hat{\boldsymbol{\gamma}}_{k2}$ are not changing.

The question arise here why we do used backfitting and not trying to fit both linear and smoother components simultaneously in one go. This is, for example, what the `gam()` function in package `mgcv` does. The answer to this is that while this will work with penalised smoothers (that is smoother using a quadratic penalty) and probably will speed up the algorithm, backfitting gives us the opportunity to try other smoothers like `loess`, cubic smoothing splines and neural networks.

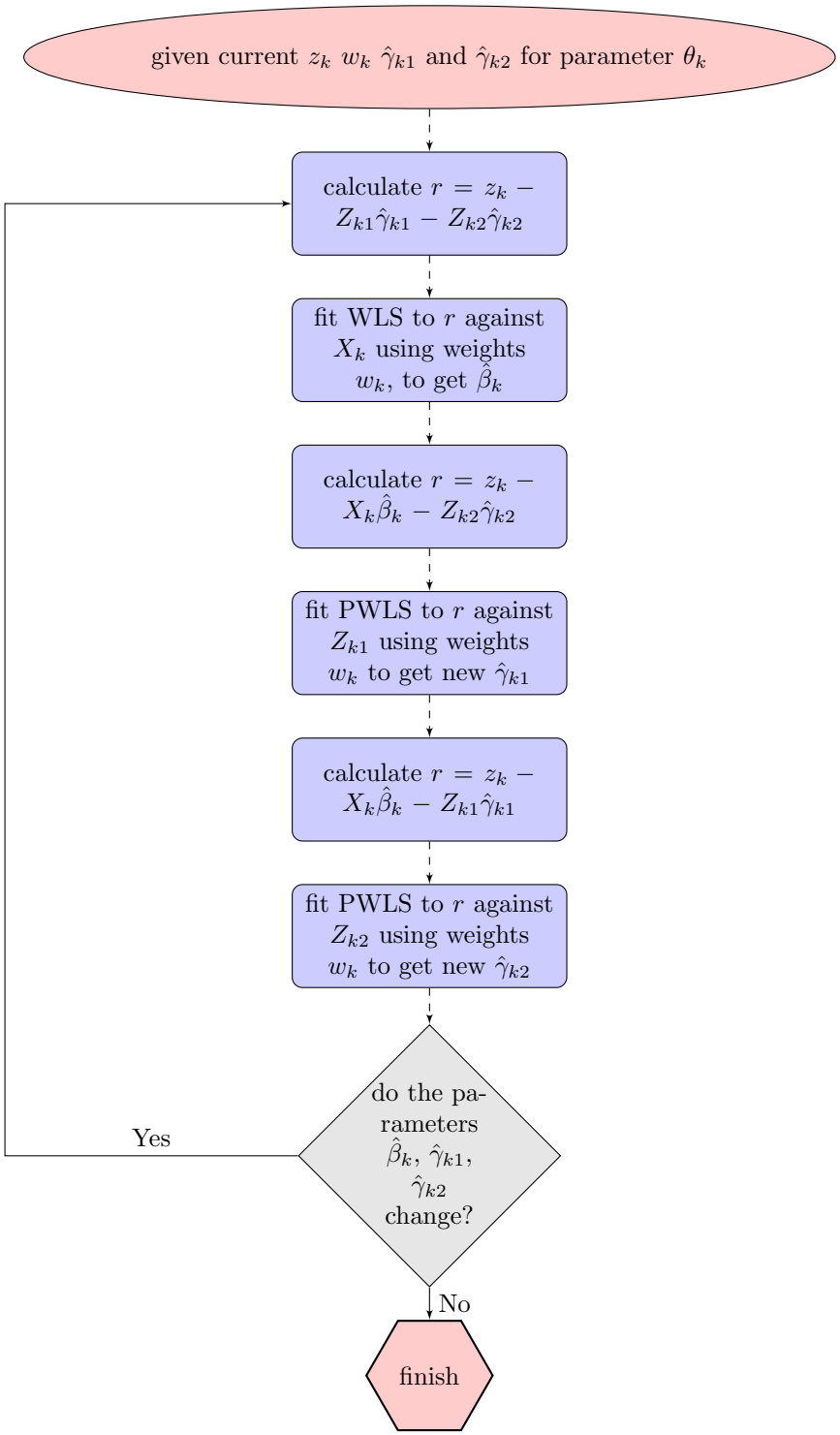


Figure 3.4: Diagram showing how the modified backfitting is working within the GAMLSS RS algorithm

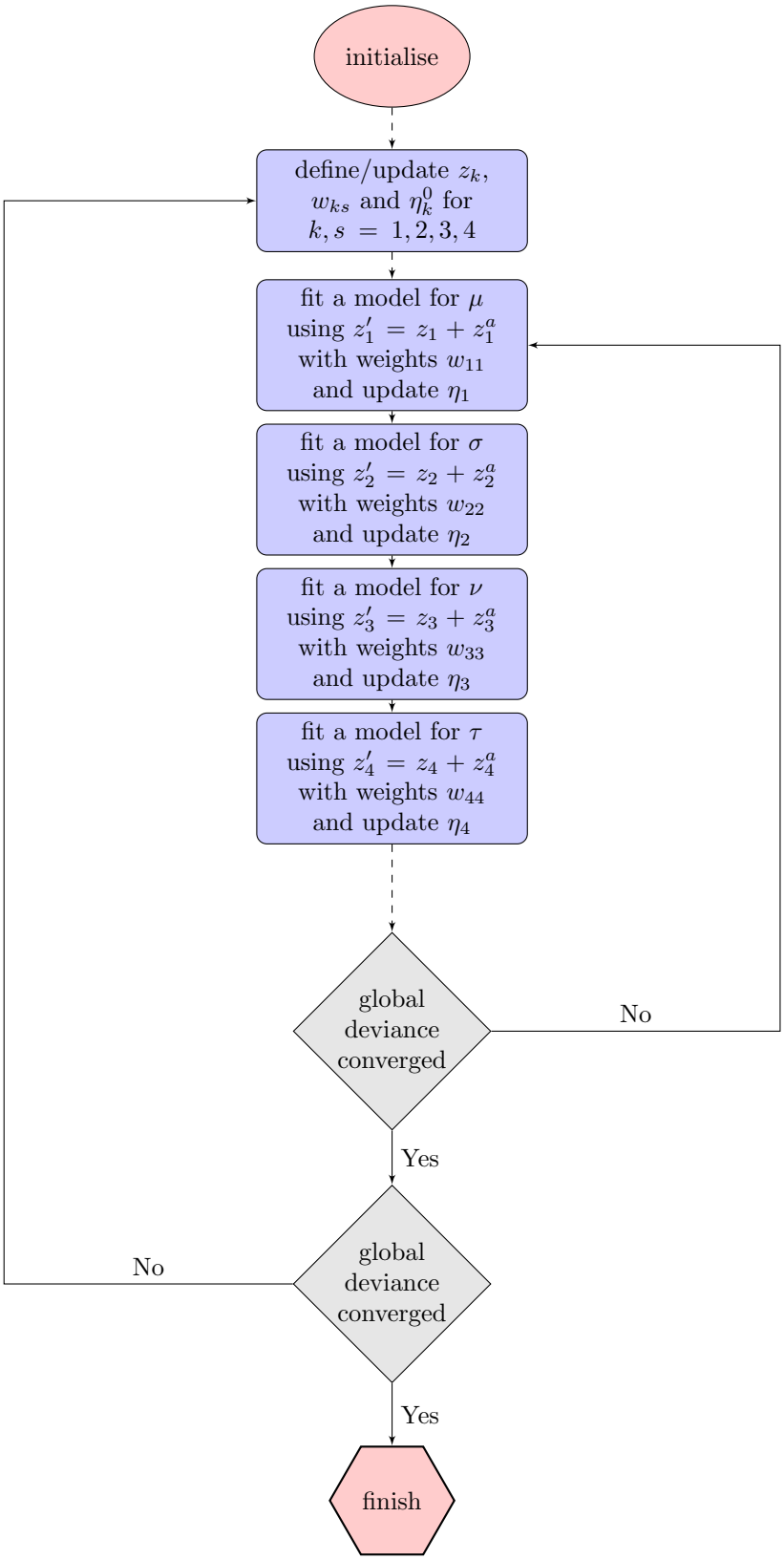


Figure 3.5: Diagram showing the outer and inner iterations within the GAMLSS CG algorithm

3.2.2 The CG algorithm

The CG algorithm is a local scoring algorithm performed within an outer and an inner iteration. Unlike the RS algorithm this algorithm needs the cross derivatives of the log likelihood with respect to each pair of parameters of the distribution. Rigby and Stasinopoulos [2005] Appendix C shows that the CG algorithm, described below, maximises the penalised likelihood (3.1) with respect to the betas, $\boldsymbol{\beta}$ and gammas, $\boldsymbol{\gamma}$ for fixed $\boldsymbol{\lambda}$.

In the outer iteration of the CG algorithm, the working variable and the iterative weights for the parameters μ, σ, ν and τ are defined by:

$$\mathbf{z}_k = \boldsymbol{\eta}_k + \mathbf{w}_{kk}^{-1} \bullet \mathbf{u}_k.$$

The $\mathbf{w}_{ks}$ vectors contain the elements of iterative weights, for $k = 1, 2, 3, 4$ and $s = 1, 2, 3, 4$, defined by $\mathbf{w}_{ks} = -\mathbf{f}_{ks} \bullet (\partial\theta_k/\partial\eta_k) \bullet (\partial\theta_s/\partial\eta_s)$ in one of three ways depending on the information available for the specific distribution:

$$\mathbf{f}_{ks} = \begin{cases} -E\left[\frac{\partial^2 \ell}{\partial \theta_k^2}\right] \\ \frac{\partial^2 \ell}{\partial \theta_k^2} \\ -\left(\right)\frac{\partial \ell}{\partial \theta_k}\Big) \bullet \left(\frac{\partial \ell}{\partial \theta_s}\right) \end{cases} \tag{3.9}$$

The inner iteration process is as follows: First it defines a new working variable as

$$\mathbf{z}_k = \mathbf{z}_k + \mathbf{z}_k^a$$

where $\mathbf{z}_k^a$ is a combination the 'cross derivatives' multiplied by the difference in the relevant predictors defined for four parameters as:

μ : $\mathbf{z}_1^a = -\mathbf{w}_{11}^{-1} \bullet [\mathbf{w}_{12} \bullet (\boldsymbol{\eta}_2 - \boldsymbol{\eta}_2^o) + \mathbf{w}_{13} \bullet (\boldsymbol{\eta}_3 - \boldsymbol{\eta}_3^o) + \mathbf{w}_{14} \bullet (\boldsymbol{\eta}_4 - \boldsymbol{\eta}_4^o)]$

σ : $\mathbf{z}_2^a = -\mathbf{w}_{22}^{-1} \bullet [\mathbf{w}_{12} \bullet (\boldsymbol{\eta}_1 - \boldsymbol{\eta}_1^o) + \mathbf{w}_{23} \bullet (\boldsymbol{\eta}_3 - \boldsymbol{\eta}_3^o) + \mathbf{w}_{24} \bullet (\boldsymbol{\eta}_4 - \boldsymbol{\eta}_4^o)]$

ν : $\mathbf{z}_3^a = -\mathbf{w}_{33}^{-1} \bullet [\mathbf{w}_{13} \bullet (\boldsymbol{\eta}_1 - \boldsymbol{\eta}_1^o) + \mathbf{w}_{23} \bullet (\boldsymbol{\eta}_2 - \boldsymbol{\eta}_2^o) + \mathbf{w}_{34} \bullet (\boldsymbol{\eta}_4 - \boldsymbol{\eta}_4^o)]$

τ : $\mathbf{z}_4^a = -\mathbf{w}_{44}^{-1} \bullet [\mathbf{w}_{14} \bullet (\boldsymbol{\eta}_1 - \boldsymbol{\eta}_1^o) + \mathbf{w}_{24} \bullet (\boldsymbol{\eta}_2 - \boldsymbol{\eta}_2^o) + \mathbf{w}_{34} \bullet (\boldsymbol{\eta}_3 - \boldsymbol{\eta}_3^o)]$

Now given the new adjusted working variables a model for each parameter is fitted using the modified backfitting algorithm. The inner iteration is continued until the global deviance does not change. Them the algorithm returns to the outer iteration which recalculates the quantities $\mathbf{z}_k$, $\mathbf{w}_{ks}$ and $\eta_k^{(o)}$ and starts the inner iteration again. The process is described at Figure 3.5. The outer iteration stops when there is no more change in the global deviance.

3.3 Estimating λ

What we have shown up to now is two algorithms, RS and CG, for estimating the parameters $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ given the hyper-parameters $\boldsymbol{\lambda}$. For fixed $\boldsymbol{\lambda}$ both methods lead to (penalised) maximum likelihood estimators for $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$. More generally it is desirable to estimate the smoothing hyper-parameters $\boldsymbol{\lambda}$ automatically. The problem now is how to estimate $\boldsymbol{\lambda}$? There are different ways of estimating the hyper-parameters $\boldsymbol{\lambda}$. Estimation can be done:

locally: when the method of estimation of each λ_{kj} is applied each time within the backfitting algorithm of the `RS` or `CG` GAMLSS algorithm

globally: when the method is applied outside the `RS` or `CG` GAMLSS algorithm.

In addition there are (at least) three different methodologies for estimating the smoothing hyper-parameters:

- Generalised cross validation (GCV),
- Generalised Akaike information criterion (GAIC), and
- Maximum likelihood based methods (ML/REML).

Table 11.1 shows where information about the different methods can be obtained.

Global	**Method**	**Reference**
Global	ML /REML (e.g. Laplace)	Rigby and Stasinopoulos [2005]
Global	GAIC (e.g. AIC, SBC)	Rigby and Stasinopoulos [2004, 2006a]
Global	Validation Global Deviance (VGD)	Stasinopoulos and Rigby [2007]
Local	ML	Rigby and Stasinopoulos [2013]
Local	GAIC	Rigby and Stasinopoulos [2013]
Local	Generalized Cross Validation (GCV)	Wood [2006]

Table 3.1: Showing references for the different approaches of choosing the smoothing parameters

In our experience the local methods are much faster and often produce similar results to the global methods. The global methods can sometimes be more reliable but they are computationally intensive. The current facilities within the GAMLSS packages allows only the global GAIC through the function `find.hyper()` and the local methods through different options when smoothers are used. For example, `pb(x)` and `pb(x, method="GAIC")` will allow using a local ML and GAIC method respectively to estimate the smoothing parameter when P-splines is used for smoothing `x`. See also Chapter ??? for more details.

All local methods assume that locally (close to the maximum) the current partial residuals, $\boldsymbol{\varepsilon}$, behave like a normally distributed random variable. Note that the 'current' refers to the fact that the partial residuals are calculated within the backfitting algorithm.

Local maximum likelihood

On the predictor scale in the $\boldsymbol{\gamma}$ fitting part of the backfitting algorithm the following (approximate) internal random effects model is assumed in order to estimate the current smoothing parameter λ:

$$\begin{aligned} \boldsymbol{\varepsilon} &= \mathbf{Z}\boldsymbol{\gamma} + \mathbf{e} \\ \mathbf{e} &\sim N(\mathbf{0}, \sigma_e^2 \mathbf{W}) \\ \boldsymbol{\gamma} &\sim N(\mathbf{0}, \sigma_b^2 \mathbf{G}^{-1}) \end{aligned} \tag{3.10}$$

Let and where $\mathbf{S}$ be the smoothing matrix so $\hat{\boldsymbol{\varepsilon}} = \mathbf{S}\boldsymbol{\varepsilon}$.

where $\boldsymbol{\varepsilon}$, are the partial residuals (within backfitting), $\mathbf{Z}$ is the basis for smoothing the current x-variable, the matrix $\mathbf{W}$ is a diagonal matrix having as values the iterative weights $\mathbf{W} = diag(\mathbf{w})$ and where $\mathbf{G}$ is a known precision matrix depending on which method for smoothing is used (see Chapter **??** for the definition of $\mathbf{G}$). The simple random effect model of equation (3.10) has the following unknown parameters to be estimated by fitting the model: σ_e^2, σ_b^2 and $\boldsymbol{\gamma}$. The smoothing parameter λ, for smoothing the explanatory variable x, is the ratio of the two variances, i.e. $\lambda = \sigma_e^2/\sigma_b^2$. The parameters σ_e^2, σ_b^2 and $\boldsymbol{\gamma}$ of model (3.10) can be estimated, see for example Rigby and Stasinopoulos [2013], using the following simple algorithm.

step 1 given the current λ estimate the $\boldsymbol{\gamma}$ parameters using a penalised least squares procedure, $\hat{\boldsymbol{\gamma}} = \left(\mathbf{Z}^\top \mathbf{W}\mathbf{Z} + \lambda \mathbf{G}\right)^{-1} \mathbf{Z}^\top \mathbf{W}\boldsymbol{\varepsilon}$

step 2 given the latest $\hat{\boldsymbol{\gamma}}$ calculate $\hat{\boldsymbol{\varepsilon}} = \mathbf{Z}\hat{\boldsymbol{\gamma}} = \mathbf{S}\boldsymbol{\varepsilon}$ where $\mathbf{S} = \mathbf{Z}\left(\mathbf{Z}^\top \mathbf{W}\mathbf{Z} + \lambda \mathbf{G}\right)^{-1} \mathbf{Z}^\top \mathbf{W}$ and compute

$\sigma_e^2 = \left(\boldsymbol{\varepsilon} - \hat{\boldsymbol{\varepsilon}}\right)^\top \left(\boldsymbol{\varepsilon} - \hat{\boldsymbol{\varepsilon}}\right) / \left(n - tr(\mathbf{S})\right)$

$\sigma_b^2 = \hat{\boldsymbol{\gamma}}^\top \hat{\boldsymbol{\gamma}} / tr(\mathbf{S})$ and therefore a new

$\hat{\lambda} = \hat{\sigma}_e^2 / \hat{\sigma}_b^2$

step 3 stop if there is no change in λ otherwise go back to step 1.

Local generalised Akaike information criterion

The local generalised Akaike information criterion (GAIC) minimises with respect to λ and for given penalty k the quantity:

$$G_{AIC} = \left|\left|\sqrt{\mathbf{w}} \bullet \left(\boldsymbol{\varepsilon} - \mathbf{Z}\hat{\boldsymbol{\gamma}}\right)\right|\right|^2 + k \times tr(\mathbf{S})$$

Hence $k = 2$ gives the local AIC and $k = \log(n)$ gives the local BIC/SBC.

Local generalised cross validation

The generalised cross validation minimise with respect to λ the quantity:

$$V_g = \frac{n \left|\left|\sqrt{\mathbf{w}} \bullet \left(\boldsymbol{\varepsilon} - \mathbf{Z}\hat{\boldsymbol{\gamma}}\right)\right|\right|^2}{\left[n - tr(\mathbf{S})\right]^2}$$

Note that by using any of the above methods to calculate locally the smoothing parameters, the `RS` or `CG` algorithms are not necessarily optimum in the the sense that will lead to the global solution. In practice though the algorithm generally work well and leads to sensible results.

3.4 Remarks on the GAMLSS algorithms

The following are general comments related to the fitting algorithms `RS` and `CG` in GAMLSS:

1. Both `RS` and `CG` algorithms can be easily implemented in any computer program which has a good weighted linear least squares algorithm.
2. The fitting procedure is a modular fitting making checking easy.
3. Additional distributions can be added easily since their contribution comes through the first and second (and optionally the cross) derivatives and therefore is orthogonal to the main algorithm.
4. The modified backfitting (Gauss-Seidel) algorithm can be easily adapted to fit any extra additive terms including terms which are not necessarily based on quadratic penalties as long as the algorithm or method used has weights.
5. Easily found starting values, requiring initial values for the $\boldsymbol{\theta} = (\mu, \sigma, \nu, \tau)$ rather than for the $\boldsymbol{\beta}$ parameters. The algorithms have generally been found to be stable and fast using very simple starting values (e.g. constants) for the $\boldsymbol{\theta}$ parameters. Default values can be changed by the user if necessary.
6. The function `nlgamlss()` in the package **gamlss.nl** provides a third algorithm for fitting **parametric** linear or non-linear GAMLSS models. However the algorithm needs starting values for all the $\boldsymbol{\beta}$ parameters, rather than $\boldsymbol{\theta} = (\mu, \sigma, \nu, \tau)$, which can be difficult for the user to choose. This method uses the `nlm()` **R** function for maximization of the likelihood, which uses numerical derivatives (if the actual derivatives are not provided).
7. For a specific data set and model, the (penalized) likelihood can potentially have multiple local maxima. This can be investigated using different starting values and has generally not been found to be a problem in the data sets analysed, possibly due to the relatively large sample sizes used.
8. Singularities in the likelihood function similar to the ones reported by Crisp and Burridge [1994] can potentially occur in specific cases within the GAMLSS framework, especially when the sample size is small. For example occasionally the scale parameter σ can go towards zero. The problem can be alleviated by appropriate restrictions on the scale parameter. For example, the link function `logS`, a shifted log link from 0.00001, does not allows values less than 0.00001 to occur.
9. Introducing local methods for estimating the smoothing hyper-parameters can sometimes make `RS` and `CG` more unstable and occasionally the global deviance increases.

Having explained how the GAMLSS algorithms are working we proceed in describing the `gamlss()` function and the objects created by its use.

Chapter 4

The gamlss() function

This chapter:

- provides an introduction to the `gamlss()` function,
- shows how the information stored in a `gamlss` class model can be explored and
- explores some of the function associated with `gamlss` class objects.

4.1 Introduction to the gamlss() function

The function `gamlss()` is the main function of the package **gamlss**. It fits a Generalized Additive Model for Location, Scale and Shape (GAMLSS). Chapters **??** and **??** shown how he function can be used. In the following sections more explanation is given on how the function can be used. Section 4.2 explains the arguments of the function and Section 4.3 shows how the functions `refit` and `update` can be used. Section 5.2 of Chapter **??** describes the components of a `gamlss` object (i.e. a fitted GAMLSS model) The profiling functions `prof.dev` and and `prof.term` are described in Section **??** of Chapter **??**.

4.2 The arguments of the gamlss() function

The usage of the function is

```
gamlss(formula = formula(data), sigma.formula = ~1,
        nu.formula = ~1, tau.formula = ~1, family = NO(),
        data = sys.parent(), 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(...), ...)
```

where the arguments of the function are defined as follows

`formula` This is a standard **R** model specification formula for the μ parameter of the distribution and it is compulsory, e.g. `y` $\sim$ `x`. Note that the formula includes in the left the response variable y.

`sigma.formula` a model formula object for the σ parameter of the distribution, e.g. $\sim$`x`.

`nu.formula` a model formula for the ν parameter of the distribution, e.g. $\sim$`x`.

`tau.formula` a model formula formula for the τ parameter of the distribution, e.g. $\sim$`x`.

`family` a `gamlss.family` object which defines the (conditional) distribution of the response variable, see Chapter **??**.

`data` a data frame containing the variables occurring in the formula (see also Section 4.2.3)

`weights` a vector of weights. Note that this argument is not equivalent to the same argument of the `glm()` or `gam()` functions. Here `weights` can be used
i) to weight out observations (with weights equal to 1 or 0), or
ii) for a weighted likelihood analysis where the contribution of the observations to the likelihood is weighted by the `weights`. Typically this is appropriate if some rows of the data are identical and the weights represent the frequencies of these rows, (see also Section 4.2.3).
Any other use of the `weights` is not recommended since this could have side effects. In particular `glm()` weights do not in general translate to `gamlss()` weights and such models should instead be fitted using `offset`(s) for the parameters μ and/or σ appropriately.

`contrasts` list of contrasts to be used for some or all of the factors appearing as variables in the parameter(s) model formula.

`method` the algorithms used for GAMLSS fitting, i.e. RS(), CG() or mixed(), see Chapter 3.

`start.from` a fitted GAMLSS model from which to take the starting values for the current model

`mu.start` vector or scalar for initial values for the location parameter μ.

`sigma.start` vector or scalar for initial values for the scale parameter σ.

`nu.start` vector or scalar of initial values for the shape parameter ν.

`tau.start` vector or scalar of initial values for the shape parameter τ.

`mu.fix` whether the μ parameter should be kept fixed at the `mu.start` value during the fitting.

`sigma.fix` whether the sigma parameter should be kept fixed at the `sigma.start` value during the fitting.

`nu.fix` whether the nu parameter should be kept fixed at the `nu.start` value during the fitting.

`tau.fix` whether the tau parameter should be kept fixed at the `tau.start` value during the fitting.

`control` Control parameters of the outer iterations of the algorithm. The default setting is the `gamlss.control` function (see below).

`i.control` this sets the control parameters of the inner iterations of the RS algorithm. The default setting is the `glim.control` function

As formulas the `gamlss()` accepts all `glm()` type formulas plus several smoothing function formulas (see Chapter 9).

Important: Note that the `na.action`, and the `subset` argument common to other statistical modelling functions such as `lm` and `glm` have been removed as arguments in the `gamlss()` function.

This is because while there is only one data set in the model there are usually up to four different model frames created (one for each distribution parameter) and therefore for consistency it is easier to apply sub-setting and na.action to the whole data set and not to the individual frames.
For subsets use `data=subset(mydata, subset=<the relevant condition>)`,
for `na.action` use `data=na.omit(mydata)`

4.2.1 The `method` argument of the `gamlss()` function

There are three different algorithms available in `gamlss()` and can be specified using the argument `method`.

RS(): The default method is the RS algorithm, which does not requires accurate starting values for μ, σ, ν and τ to ensure convergence (the default starting values, often constants, are usually adequate). This method is more stable in the initial stage of the fitting and faster for larger data sets.

CG() The CG algorithm, which can be better for distributions with potentially highly correlated parameter estimates but which is very unstable in the beginning of the process.

mixed(): This is a mixture of the above two algorithms which starts with the RS algorithm and finishes with the CG.

Note that the default value of the argument `method` is the RS algorithm because of its stability.

The `RS()` and `CG()` algorithms are explained in detail in Rigby and Stasinopoulos [2005].

R data file: abdom in package **gamlss.data** of dimensions 610×2

variables

`y` : abdominal circumference

`x` : gestational age

purpose: to demonstrate the fitting of a simple regression type model in GAMLSS

For example here we use the `abdom` data , kindly provided by Dr. Eileen M. Wright, with response variable the abdominal circumference, `y` and explanatory variable the gestational age in weeks `x`. The data comprises 610 observations.

```
data(abdom)
h<-gamlss(y~pb(x), sigma.fo=~pb(x), family=NO, data=abdom)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 4786.697
## GAMLSS-RS iteration 2: Global Deviance = 4785.695
## GAMLSS-RS iteration 3: Global Deviance = 4785.696
```

fits the model using the RS algorithm. Note that the global deviance can increase slightly during the iterations. This can happen if smoothing additive terms are involved since the degrees of freedom in the different fits could change very slightly. The CG algorithm is used by:

```
h<-gamlss(y~pb(x), sigma.fo=~pb(x), family=NO, data=abdom,
            method=CG())
```

```
## GAMLSS-CG iteration 1: Global Deviance = 6165.522
## . . .
## GAMLSS-CG iteration 9: Global Deviance = 4785.695
```

and the mixed algorithm is used by:

```
h<-gamlss(y~pb(x), sigma.fo=~pb(x), family=NO, data=abdom,
            method=mixed(2,20))
```

```
## GAMLSS-RS iteration 1: Global Deviance = 4786.697
## GAMLSS-RS iteration 2: Global Deviance = 4785.695
## GAMLSS-CG iteration 1: Global Deviance = 4785.696
```

In the above example the mixed method uses 2 cycles of the RS algorithm, followed by up to 20 cycles of the CG algorithm. All methods end up essentially with the same fitted model, a useful check.

4.2.2 The algorithmic control functions

The `gamlss.control` function is defined as

```
gamlss.control(c.crit = 0.001, n.cyc = 20, mu.step = 1, sigma.step = 1, nu.step = 1,
               tau.step = 1, gd.tol = 5, iter = 0, trace = TRUE, ...)
```

where

`c.crit` is the convergence criterion for the outer iteration of the algorithms

`n.cyc` is maximum number of cycles of the outer iteration of the algorithms

`mu.step` is the inner iteration step length for the parameter μ

`sigma.step` is the inner iteration step length for the parameter σ

`nu.step` is the inner iteration step length for the parameter ν

`tau.step` is the inner iteration step length for the parameter τ

`gd.tol` global deviance tolerance level, this allows the global deviance to temporarily increase useful if fitting complicate models with a lot of smoothing parameters.

`iter` this should not normally be used by the user. It is used when the (refit) function is used to count the right number of iterations

`trace` whether to print the global deviance at each outer iteration of the RS() and CG() algorithms. The users are advised to keep the default values TRUE so they can check if the algorithm is converging properly.

The function which controls parts of the inner iteration is `glim.control`

```
glim.control(cc = 0.001, cyc = 50,  trace = FALSE, bf.cyc = 30, bf.tol = 0.001,
             bf.trace = FALSE,...)
```

where

`cc` is the convergence criterion for the inner iteration or GLIM part of algorithm

`cyc` the number of cycles of the inner iteration GLIM part of the algorithm

`trace` whether to print at each inner iteration of the GLIM part of the algorithm with default FALSE.

`bf.cyc` the number of cycles of the backfitting algorithm (see section)

`bf.tol` the convergence criterion (tolerance level=10^{-3} by default) for the backfitting algorithm see 3.2.1

`bf.trace` whether to print at each iteration of the backfitting (TRUE) or not (FALSE, the default).

Here is an example of how to change the convergence criterion `c.crit`. First fit the model with the default convergence criterion value of 0.001.

```
h<-gamlss(y~pb(x), sigma.fo=~pb(x), family=NO, data=abdom)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 4786.697
## GAMLSS-RS iteration 2: Global Deviance = 4785.695
## GAMLSS-RS iteration 3: Global Deviance = 4785.696
```

Now change the convergence criterion to 0.000001 using `control` argument in `gamlss()` with the criterion defined within `gamlss.control()`.

```
h<-gamlss(y~pb(x), sigma.fo=~pb(x), family=NO, data=abdom, c.crit=0.000001)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 4786.697
## . . .
## GAMLSS-RS iteration 5: Global Deviance = 4785.696
```

Now let us change the default values of the `trace` option of in the `i.control` argument defined within `glim.control()`.

```
h<-gamlss(y~pb(x), sigma.fo=~pb(x), family=NO, data=abdom, glm.trace=TRUE)
```

```
## GLIM iteration 1 for mu: Global Deviance = 6607.265
## . . .
## GLIM iteration 1 for sigma: Global Deviance = 6036.217
```

This trick is useful when checking the convergences for the individual distribution parameters but, unless a problem is suspected, it is better to leave it at the default value.

Useful Advice: If a large data set is used (say more than 10000 observations), and the user is at an exploitative stage of the analysis, where many models have to be fitted relatively fast, it is advisable to change the `c.crit` in `gamlss.control()` to something like 0.01 or even 0.1.

Let us now fit the t distribution to the above data. The `family` option for the t distribution family is `TF` and the t distribution degrees of freedom parameter is `nu` and is fitted as constant by default.

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

```
## GAMLSS-RS iteration 1: Global Deviance = 4780.234
## GAMLSS-RS iteration 2: Global Deviance = 4777.493
## GAMLSS-RS iteration 3: Global Deviance = 4777.519
## GAMLSS-RS iteration 4: Global Deviance = 4777.52
```

The fitted value for the constant degrees of freedom parameter `nu` is 11.42 and can be obtained using `fitted(h,"nu")[1]` or `exp(coef(h,"nu"))`. There are occasions where the user wants to fix the parameter(s) of a distribution at specific value(s). For example, one might want to fix the degrees of freedoms of the t distribution say at 10. This can be done as follows with the `nu.start` and `nu.fix` arguments:

```
h1<-gamlss(y~pb(x), sigma.fo=~pb(x), family=TF, data=abdom, nu.start=10,
                  nu.fix=TRUE)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 4780.35
## GAMLSS-RS iteration 2: Global Deviance = 4777.61
## GAMLSS-RS iteration 3: Global Deviance = 4777.633
## GAMLSS-RS iteration 4: Global Deviance = 4777.633
```

Note The t distribution may be unstable if ν is fixed close to one (usually indicating that this is an inappropriate value of ν for the particular data set).

4.2.3 Weighting out observations, the weights and data=subset() arguments

There are two ways in which the user can weight out observations from the analysis. The first relies on the `subset()` function of **R** and can be used in the `data` argument of `gamlss()`, i.e. `data=subset(mydata, condition)`, where `condition` is a relevant `R` code restricting the case numbers of the data.

Important: It was mentioned earlier that the `subset` argument of `lm()` and `glm()` functions is not an argument in `gamlss()`. Always use `data=subset(mydata, condition)`.

The second way is through the `weights` option. Note that the `weights` are not performing in the same way as in the `glm()` or `lm()` functions. There they are prior weights used to fit only the mean of the model, while here the same weights are applied for fitting all (possibly four) parameters. The `weights` here can be used for a weighted likelihood analysis where the contribution of the observations to the log likelihood is weighted according to `weights`. Typically this is appropriate in the following cases:

frequencies: if some rows of the data are identical and the weights represent the frequencies of these rows

zero weights: A more common application of the weights is to set them equal to zero or one (i.e. FALSE or TRUE), so observations can be weighted out from the analysis

weighted log-likelihood: This is the case where different weights in the log-likelihood for different observations is required. One example is the use of `gamlss` objects in the fitting of finite mixtures, see package `gamlss.mx`.

Note than in general a model fitted to the original uncollapsed `data.frame` or to the collapsed `data.frame` using frequencies as weights should produce identical results in terms of fitted model parameters. The fitted values and the residuals of the two different models do not have to have the same length as we will demonstrate in this Section.

Note that using `data=subset()` only fits the data cases in the subset, so fitted values for the parameters are only calculated for the subset data cases. However using the weights option fits all the data cases (although cases with weights 0 do not contribute to the fit) and so fitted values for the parameters are calculated for all data cases. These fitted values will be correct and this is a method to produce predictive values the other is using the function `predict()`].

Let us assume that in our abdominal circumference example we want to weight out all observations in which the `x` variable is less than or equal to 20. We can do this using the function `subset()`.

```
h2<-gamlss(y~pb(x), sigma.fo=~pb(x), family=TF, data=subset(abdom,x>20))
```

```
## GAMLSS-RS iteration 1: Global Deviance = 3706.584
## . . .
## GAMLSS-RS iteration 4: Global Deviance = 3706.763
```

```
c(length(fitted(h2)), length(resid(h2)), h2$noObs, h2$N)
```

```
## [1] 456 456 456 456
```

Note that `h2$N` gives the length of the response variable while `h2$noOBS` is the sum of the weights. Now we use `weights`:

```
h3<-gamlss(y~pb(x), sigma.fo=~pb(x), family=TF, data=abdom, weights=x>20)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 3706.698
## . . .
## GAMLSS-RS iteration 4: Global Deviance = 3706.827
```

```
c(length(fitted(h3)), length(resid(h3)), h3$noObs, h3$N)
```

```
## [1] 610 456 456 610
```

Let us assume now that we want to weight out only a few observations, say the 200th and 400th. We can do it neither way using `subset` or `weights`. The advantage of using the argument `weights` is that we can get predictions for those values:

```
w <- rep(1, 610)
w[c(200,400)] <-  0
h41<-gamlss(y~pb(x), sigma.fo=~pb(x), family=TF,
            data=subset(abdom, w==1))
```

```
## GAMLSS-RS iteration 1: Global Deviance = 4766.151
## GAMLSS-RS iteration 2: Global Deviance = 4763.481
## GAMLSS-RS iteration 3: Global Deviance = 4763.506
## GAMLSS-RS iteration 4: Global Deviance = 4763.507
```

```
h42<-gamlss(y~pb(x), sigma.fo=~pb(x), family=TF, weights=w,
            data=abdom)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 4766.151
## GAMLSS-RS iteration 2: Global Deviance = 4763.481
## GAMLSS-RS iteration 3: Global Deviance = 4763.506
## GAMLSS-RS iteration 4: Global Deviance = 4763.507
```

```
fitted(h42, "mu")[c(200,400)]
```

```
## [1] 176.8339 278.3580
```

If the variables in the reduced `data.frame` are to be used extensively later on, it would make more sense to use the `subset` function in advance of the fitting to create a reduced data set.

The following simple artificial example demonstrates the use of the `weights` argument when frequencies are involved in the data. [The approach is particularly suited to fitting discrete distributions to frequency count data.]

```
  y <- c(3,3,7,8,8,9,10,10,12,12,14,14,16,17,17,19,19,18,22,22 )
  x <- c(1,1,2,3,3,4, 5, 5, 6, 6, 7, 7, 8, 9, 9,10,10,11,12,12 )
ex1 <- data.frame(y=y,x=x)
```

The 20×2 data frame `ex1` contains some identical rows, e.g. row 1 and 2 or 7 and 8. A new

data frame, containing the same information as in `ex1`, but with an extra variable called `freq` indicating the number of identical rows in `ex1` in can be create as:

```
yy <- c(3, 7, 8, 9, 10, 12, 14, 16, 17, 19, 18, 22)
xx <- c(1, 2, 3, 4,  5,  6,  7,  8,  9, 10, 11, 12)
ww <- c(2, 1, 2, 1,  2,  2,  2,  1,  2,  2,  1,  2)
ex2 <- data.frame(y=yy, x=xx, freq=ww)
```

Fitting a statistical model using each of the two data frames should produce identical results. This is demonstrated below where prior `weights` are used to fit the data in `ex2`.

```
m1 <- gamlss(y~x, data=ex1, family=PO)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 90.8238
## GAMLSS-RS iteration 2: Global Deviance = 90.8238
```

```
m2 <- gamlss(y~x,  weights=freq, data=ex2, family=PO)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 90.8238
## GAMLSS-RS iteration 2: Global Deviance = 90.8238
```

```
all.equal(deviance(m1),deviance(m2))
```

```
## [1] TRUE
```

```
c(length(fitted(m1)), length(resid(m1)), m1$noObs,  m1$N)
```

```
## [1] 20 20 20 20
```

```
c(length(fitted(m2)), length(resid(m2)), m2$noObs, m2$N)
```

```
## [1] 12 20 20 12
```

Note the lengths of the fitted values and the residuals of the two models. In the case of model `m2` the residuals are expanded to represent all 20 original observations. Note that `resid(m1)` and `resid(m2)` are not going to be identical in this case since both are randomized residuals due to the fact we used a discrete distribution.

The user may be tempted to scale the weights but this may have undesirable consequences as we demonstrate below.

```
m3<- gamlss(y~x, sigma.fo=~x,  weights=freq/2, data=ex2, family=PO)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 45.4119
## GAMLSS-RS iteration 2: Global Deviance = 45.4119
```

```
summary(m2)
```

```
##
## . . .
## Mu link function:  log
## Mu Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.62331    0.16548   9.809 1.20e-08 ***
## x            0.12904    0.01914   6.741 2.56e-06 ***
## ---
## . . .
```

```
summary(m3)
```

```
##
## . . .
## ------------------------------------------------------------------
## Mu link function:  log
## Mu Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.62331    0.23403   6.936 4.01e-05 ***
## x            0.12904    0.02707   4.767 0.000761 ***
## . . .
```

```
deviance(m2)
```

```
## [1] 90.82379
```

```
deviance(m3)
```

```
## [1] 45.41189
```

We can see that, while in this specific example the fitted coefficients are the same, the deviances and more importantly the standard errors have been affected by the change in `weights`. Also because the weights are not frequencies the length of the residuals remains 12. In general using weights that are not frequencies is **not** recommended unless the user knows what he/she doing and is aware of the problem.

4.3 The refit and update functions

4.3.1 refit()

The function `refit()` can be used if the `converged` component of the `gamlss` fitted object is FALSE, that is, when the maximum number of iteration `c.cyc` has been reached without convergence. The default value for `c.cyc` is 20 and it is usually sufficient for most problems. Here it is an artificial example in which we force the algorithm to stop in the second iteration so we can continue with `refit()`

```
h<-gamlss(y~pb(x), sigma.fo=~pb(x), family=TF, data=abdom, n.cyc=3 )
```

```
## GAMLSS-RS iteration 1: Global Deviance = 4780.234
## GAMLSS-RS iteration 2: Global Deviance = 4777.493
## GAMLSS-RS iteration 3: Global Deviance = 4777.519
```

```
## Warning in RS():  Algorithm RS has not yet converged
```

```
h<-refit(h)
```

```
## GAMLSS-RS iteration 4: Global Deviance = 4777.52
```

4.3.2 update()

The function `update()` can be used to update formulae or other arguments of a `gamlss` fitted object. To update formulae `update` uses the the **R** `update.formula()` function to update the specified distribution parameter.

The **gamlss** `update()` function is defined as

```
update.gamlss(object, formula., ..., what = c("mu", "sigma", "nu", "tau"),
              evaluate = TRUE)
```

where

`object`	a `gamlss` fitted object
`formula`	the formula to update
`...`	for updating argument in `gamlss()`
`what`	what parameter of the distribution is required for updating in the formula, `mu`, `sigma`, `nu` or `tau`, the default is `what="mu"`
`evaluate`	whether to evaluate the call or not (the default is TRUE)

R data file: aids in package **gamlss.data** of dimensions 45×3

variables

`y` : the number of quarterly aids cases in England and Wales

`x` : time in quarters from January 1983 [item[`qrt`] : a factor for the quarterly seasonal effect

purpose: to demonstrate the fitting of a simple regression type model in GAMLSS

Here we use the `aids` data which consist of 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. We start by using the Poisson family to model the number of reported cases (the response variable), against time (a continuous explanatory variable) which we smooth with a cubic spline smoother using 5 effective degrees of freedom and against `qrt` a factor representing quarterly seasonal effect. We then (i) change the family to negative binomial (type I) (ii) update the smoothing parameter with `df=8` (iii) remove the quarterly seasonal effect (iv) and finally fit a normal family model with response the `log(y)`.

```
data(aids)
# fit a poisson model
h.po <-gamlss(y~pb(x)+qrt, family=PO, data=aids)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 387.1462
## . . .
## GAMLSS-RS iteration 3: Global Deviance = 387.1547
```

```
# update with a negative binomial
 h.nb <-update(h.po, family=NBI)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 373.1785
## . . .
## GAMLSS-RS iteration 5: Global Deviance = 366.9258
```

```
# update the smoothing using cs()
 h.nb1 <-update(h.nb,~cs(x,8)+qrt)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 362.9323
## . . .
## GAMLSS-RS iteration 5: Global Deviance = 359.2348
```

```
# remove qrt
  h.nb2 <-update(h.nb1,~.-qrt)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 379.5915
## . . .
## GAMLSS-RS iteration 4: Global Deviance = 379.5626
```

```
 # put back qrt take log of y and fit a normal distribution
h.nb3 <-update(h.nb1,log(.)~.+qrt, family=NO)

## GAMLSS-RS iteration 1: Global Deviance = -42.3446
## GAMLSS-RS iteration 2: Global Deviance = -42.3446

 # verify that it is the same
 h.no<-gamlss(log(y)~cs(x,8)+qrt,data=aids )

## GAMLSS-RS iteration 1: Global Deviance = -42.3446
## GAMLSS-RS iteration 2: Global Deviance = -42.3446
```

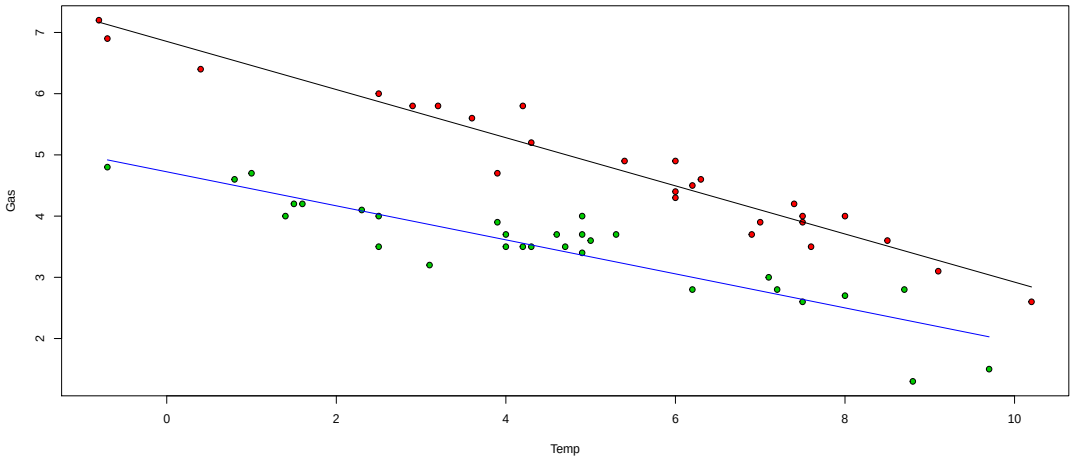
Finally we give an example taken from see Venables and Ripley [2002] Section 6.1, to demonstrate how update can be used to fit two different lines in a analysis of covariance situation. Each model fits a separate regression of gas consumption on temperature for the two different levels of the factor `Insul`. The data are gas consumtion, `Gas`, the average outside temperature in degrees Celsius, `Temp`, and `Insul` a factor, before or after insulation.

```
library(MASS)
data(whiteside)
gasB <- gamlss(Gas~Temp, data=subset(whiteside, Insul=="Before"))

## GAMLSS-RS iteration 1: Global Deviance = 5.7566
## GAMLSS-RS iteration 2: Global Deviance = 5.7566

gasA <- update(gasB,data=subset(whiteside, Insul=="After"))

## GAMLSS-RS iteration 1: Global Deviance = 20.9026
## GAMLSS-RS iteration 2: Global Deviance = 20.9026
```

Figure 4.1 shows the gas consumption against the average outside temperature in degrees Celsius for before or after insulation.

Figure 4.1

```
with(whiteside, plot(Temp,Gas,pch=21,bg=c("red","green3")[unclass(Insul)]))
with(whiteside, lines(Temp[Insul=="Before"],fitted(gasB)))
with(whiteside, lines(Temp[Insul=="After"],fitted(gasA), col="blue"))
```



R code on page **??**

Figure 4.1: A linear interaction model for gas consumption against the average outside temperature in degrees Celsius for before or after insulation

Chapter 5

Methods for fitted gamlss objects

This chapter:

- provides information how to deal with a gamlss fitted object
- shows how several methods applied to gamlss objects are working including predict() and summary()
- provides information for the use of the inferential functions prof.dev() and prof.term()

5.1 Introduction

This Chapter describes methods, (that is, **R** functions) which can be used to display information from a fitted gamlss object. The gamlss() function creates S3 class objects. Note that not all methods for a gamlss object are described here. For example, methods associated with additive terms, diagnostics and selection of variables are described in Chapter 8, 12 and **??** respectively. The following methods are considered in this chapter.

deviance() for extracting the global deviance

edf(), edfAll() for extracting the effective degrees of freedom

fitted(), fv() for extracting the fitted values

formula() for extracting a model formula

gen.likelihood() for generating the likelihood function of the fitted model (used by vcoc())

get.K() for extracting the K matrix (meat) for sandwich standard errors

getSmo() for extracting information from a fitted smoothing term

logLik() for extracting the log likelihood

`lp()` for extracting the linear predictor for a distribution parameter (see also `lpred`)

`lpred()` for extracting the fitted values, linear predictor or specified terms (with standard errors) for a distribution parameter.

`model.frame()` for extracting the model frame of a specified distribution parameter

`model.matrix()` to extract the design matrix of a specified distribution parameter

`predict()`, `predictAll()` to predict from new data individual distribution parameter values (see also `lpred` above)

`print()` : for printing a `gamlss` object

`residuals()` to extract the normalised (randomised) quantile residuals from a fitted `gamlss` model object, see Chapter 12.

`Rsq()` for getting the generalised R-squared

`rvcov()` for extracting the robust (sandwich) variance-covariance matrix of the beta estimates (for all distribution parameter models). It can be done also with `vcov()`.

`summary()` to summarize the fit in a `gamlss` object

`terms()` for extracting terms from a `gamlss` object

`vcov()` to extract the variance-covariance matrix of the beta estimates (for all distribution parameter models).

5.2 The gamlss object

The function `gamlss()` returns a `gamlss` S3 object, that is, a GAMLSS fitted model which may have converged or not depending whether the maximum number of iterations given by `c.cyc` has been reached or not.

The generic functions `print` and `summary` can be used to print and summarise the object as was indicated in Chapter 2. The model

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

```
## GAMLSS-RS iteration 1: Global Deviance = 4780.234
## . . .
## GAMLSS-RS iteration 4: Global Deviance = 4777.52
```

is used here to demonstrate the composition of a gamlss object. By calling the `names` function we are able to check on the components of the object `h`.

```
names(h)
```

```
##  [1] "family"            "parameters"        "call"
##  [4] "y"                 "control"           "weights"
##  [7] "G.deviance"        "N"                 "rqres"
## [10] "iter"              "type"              "method"
## [13] "contrasts"         "converged"         "residuals"
## [16] "noObs"             "mu.fv"             "mu.lp"
```

```
## [19] "mu.wv"              "mu.wt"              "mu.link"
## [22] "mu.terms"           "mu.x"               "mu.qr"
## [25] "mu.coefficients"    "mu.offset"          "mu.xlevels"
## [28] "mu.formula"         "mu.df"              "mu.nl.df"
## [31] "mu.s"               "mu.var"             "mu.coefSmo"
## [34] "mu.lambda"          "mu.pen"             "df.fit"
## [37] "pen"                "df.residual"        "sigma.fv"
## [40] "sigma.lp"           "sigma.wv"           "sigma.wt"
## [43] "sigma.link"         "sigma.terms"        "sigma.x"
## [46] "sigma.qr"           "sigma.coefficients" "sigma.offset"
## [49] "sigma.xlevels"      "sigma.formula"      "sigma.df"
## [52] "sigma.nl.df"        "sigma.s"            "sigma.var"
## [55] "sigma.coefSmo"      "sigma.lambda"       "sigma.pen"
## [58] "nu.fv"              "nu.lp"              "nu.wv"
## [61] "nu.wt"              "nu.link"            "nu.terms"
## [64] "nu.x"               "nu.qr"              "nu.coefficients"
## [67] "nu.offset"          "nu.formula"         "nu.df"
## [70] "nu.nl.df"           "nu.pen"             "P.deviance"
## [73] "aic"                "sbc"
```

More generally any `gamlss` object has the following components

`family` The distribution family of the `gamlss` object (see Chapter **??**) e.g. for the object `h` we have

```
 h$family
```

```
## [1] "TF"       "t Family"
```

`parameters` The name of the fitted parameters as a character list.

```
 h$parameters
```

```
## [1] "mu"    "sigma" "nu"
```

`call` The call of the `gamlss()` function e.g.

```
 h$call
```

```
## gamlss(formula = y ~ pb(x), sigma.formula = ~pb(x), family = TF,
##     data = abdom)
```

`y` The response variable as a vector (or matrix), accessed by h$y

`control` The `gamlss()` fit control settings, accessed by h$control

`weights` The vector of prior weights, accessed by h$weights

`G.deviance` The value of global deviance which can be extracted by h$G.deviance or by using the generic function `deviance()` or `deviance(gamlss.object, "G")` e.g.

```
deviance(h)
```

```
## [1] 4777.519
```

`N` The length of the response variable (or the number of observations in the fit unless `weights` are used) e.g.

```
h$N
## [1] 610
```

`noObs` The actual number of observations if weights are used (e.g.. to weight out observations) in the fit equal to the sum of the weights. If no `weights` are used is equal to `h$N`.

```
h$noObs
## [1] 610
```

`rqres` A function to calculate the (normalized randomized quantile) residuals of the object, accessed by h$rqres. [The residuals are randomized for discrete distributions only, see Dunn and Smyth [1996]]

`iter` The number of external iterations in the fitting process (by an external iteration we mean the refitting of all the distribution parameters μ, σ, ν and τ), i.e.

```
h$iter
## [1] 4
```

`type` The type of the distribution of the response variable (continuous, discrete or mixed) i.e.

```
h$type
## [1] "Continuous"
```

`method` Which algorithm is used for the fit, RS(), CG() or mixed() i.e.

```
h$method
## RS()
```

`contrasts` Which contrasts were used in the fit, NULL if they have not been set in `gamlss()` function

`converged` Whether the model has converged i.e.

```
 h$converged
## [1] TRUE
```

`residuals` The (normalized randomized quantile) residuals of the model which can be extracted by `h$residuals` or by using the generic function `residuals()`, (also abbreviated as `resid()`). [These residuals are randomized for discrete distributions only. See Dunn and Smyth (1996) or Chapter 12.]

`df.fit` The total degrees of freedom use by the model, e.g. in the model `h` there are 3 distribution parameters μ and σ and ν. The total degrees of freedom for the fit is the summation of all the degrees of freedom used to fit the individual

parameters. Those degrees of freedom are stored in `h$mu.df`, `h$sigma.df` and `h$nu.df` respectively.

```
h$df.fit

## [1] 8.789107

h$mu.df+h$sigma.df+h$nu.df

## [1] 8.789107
```

`df.residual` The residual degrees of freedom left after the model is fitted

```
h$df.residual

## [1] 601.2109
```

`pen` The sum of the quadratic penalties for all the parameters (if appropriate additive terms are fitted)

```
h$pen

## [1] 3.839226
```

`P.deviance` The penalized deviance, Global deviance + penalties, which can be extracted by `h$P.deviance` or by using the generic function `deviance(gamlss.object,"P")` e.g.

```
h$P.deviance

## [1] 4781.359

deviance(h,"P")

## [1] 4781.359
```

`aic` The Akaike information criterion, which can be also obtained using the functions `AIC()` or `GAIC()`.

```
h$aic

## [1] 4795.098

AIC(h)

## [1] 4795.098

GAIC(h)

## [1] 4795.098
```

`sbc` The Bayesian information criterion (BIC) or the Schwartz Bayesian criterion (SBC), which can be also extracted using `AIC()` or `GAIC()`.

```
h$sbc

## [1] 4833.888

AIC(h, k=log(length(abdom$y)))

## [1] 4833.888
```

```
GAIC(h, k=log(length(abdom$y)))
## [1] 4833.888
```

The rest of the components refer to the parameters of the model (if they exist). The name `par` below should be replaced with the appropriate parameter, which can be any of the `mu`, `sigma`, `nu` or `tau`).

`par.fv` The fitted values of the appropriate parameter accessed by e.g. `h$mu.fv`. The fitted values can also be extracted using the generic function `fitted()` e.g. `fitted(h,"mu")` extracts the `mu` fitted values while `fitted(h,"sigma")` extracts the `sigma` fitted values.

```
head(fitted(h)) # equivalent to fitted(h,"mu")
## [1] 60.81081 60.81081 60.81081 62.58732 66.13772 66.13772
tail(fitted(h, "sigma"))
## [1] 20.46207 20.46207 20.46207 20.46207 20.58861 20.70743
```

`par.lp` The (linear) predictor of the appropriate parameter, accessed by e.g.

```
head(h$mu.lp)
## [1] 60.81081 60.81081 60.81081 62.58732 66.13772 66.13772
```

`par.wv` The working variable of the appropriate parameter.

`par.wt` The working weights of the appropriate parameter.

`par.link` The link function for appropriate parameter i.e.:

```
h$sigma.link
## [1] "log"
```

`par.terms` The terms for the appropriate parameter model.

`par.x` The design matrix for the appropriate parameter.

`par.qr` The QR decomposition of the appropriate parameter model.

`par.coefficients` The linear coefficients of the the appropriate parameter model which can also be extracted using the generic function `coef()`.

```
coef(h, "mu")
## (Intercept)       pb(x)
##   -55.61858    10.34939
```

`par.formula` The formula for the appropriate parameter model.

```
h$mu.formula
## y ~ pb(x)
```

`par.df` The appropriate parameter degrees of freedom (see obove).

```
h$mu.df
## [1] 5.787446
```

`parameter.ml.df` The non linear (e.g. smoothing) degrees of freedom for the appropriate parameter. Note that this does not include the fitted constant and linear part.

```
h$mu.nl.df
## [1] 3.787446
```

`par.pen` The sum of the quadratic penalties for the specific parameter (if appropriate additive terms are fitted).

`par.xlevels` (only where relevant) a record of the levels of the factors used in fitting of this parameter.

5.3 The `predict()`, `predictAll()` and `lpred()` functions

The function `predict.gamlss()` is the GAMLSS specific method which produces predictors for the current or a new data set for a specified parameter of a `gamlss` object. The `predict.gamlss()` can be used to extract (linear) predictors, (`type="link"`), fitted values, (`type="response"`) and contributions of terms in the (linear) predictor, (`type="terms"`), for a specific parameter in the model at the current or new values of the x-variables in a similar way that the `predict.lm()` and `predict.glm()` functions can be used for `lm` and `glm` objects. Problems associated with the above functions, see Venables and Ripley [2002] Section 6.4, are avoided here since the `predict()` function for **gamlss** is based on the safe `predict.gam()` `S-PLUS` function of Trevor Hastie, see Chambers and Hastie [1992]. Note that the main difference between the **gamlss** `predict()` and the usual predictive functions in R is the fact that the **gamlss** `predict()` function is distribution parameter specific, that is, predictions are for one of the distribution parameters `mu`, `sigma`, `nu` and `tau`.

Linear predictors, fitted values and specific terms for a specific distribution parameter in the model at the current data values of the explanatory variables can be also extracted using the function `lpred()` (which in fact is called from `predict()` if the `newdata` argument is NULL, see below).

The **gamlss** `predict()` function is defined as

```
predict(object, what = c("mu", "sigma", "nu", "tau"),
                parameter= NULL,
                newdata = NULL, type = c("link", "response", "terms"),
                terms = NULL, se.fit = FALSE, data = NULL, ...)
```

where

`object` a `gamlss` fitted object

`what` or `parameter`
what parameter of the distribution is required, `mu`, `sigma`, `nu` or `tau`, the default is `what="mu"`

`newdata`	a data frame containing new values for the explanatory variables used in the model
`type`	The default value is `type="link"` gets the (linear) predictor for the specified distribution parameter. `type="response"` gets the fitted values for the parameter and finally `type="terms"` gets the contribution of fitted terms for the specific parameter.
`terms`	if `type="terms"` is defined then this option selects the specified term from the formula of the parameter at hand. By default all terms are selected.
`se.fit`	if TRUE the approximate standard errors of the appropriate type are extracted. Note that standard errors are not given for new data sets, i.e. when `newdata` is defined.
`data`	the data frame used in the original fit if is not defined in the call
`...`	for extra arguments

The `lpred` function of **gamlss** has identical arguments to the `predict.gamlss()` function apart from the `newdata` argument which does not exist in `lpred`. The functions `fitted()` and `fv()` are equivalent to using `lpred()` or `predict()` with the argument `type="response"`. The function `lp()` is equivalent of using `lpred()` or `predict()` with the argument `type="link"`. The following code demonstrates some of the points.

```
data(aids)
 # fitting a negative binomial type I distribution
aids.1<-gamlss(y~poly(x,3)+qrt, family=NBI, data=aids) #
```

```
## GAMLSS-RS iteration 1: Global Deviance = 383.4541
## . . .
## GAMLSS-RS iteration 4: Global Deviance = 381.7145
```

```
head(predict(aids.1))
```

```
##        1        2        3        4        5        6
## 1.322524 1.490931 1.996051 2.140244 2.540856 2.643345
```

```
identical(predict(aids.1),predict(aids.1, parameter="mu"))
```

```
## [1] TRUE
```

```
identical(predict(aids.1),predict(aids.1, parameter="mu", type="link"))
```

```
## [1] TRUE
```

```
identical(predict(aids.1),lpred(aids.1))
```

```
## [1] TRUE
```

```
identical(predict(aids.1),lpred(aids.1, parameter="mu"))
```

```
## [1] TRUE
```

```
identical(predict(aids.1),lpred(aids.1, parameter="mu", type="link"))
```

```
## [1] TRUE
```

```
identical(predict(aids.1),lp(aids.1))
```

```
## [1] TRUE
```

```
identical(predict(aids.1),lp(aids.1,parameter="mu"))
```

```
## [1] TRUE
```

```
identical(predict(aids.1),lp(aids.1,parameter="mu"))
```

```
## [1] TRUE
```

```
head(predict(aids.1, type="response"))
```

```
##         1         2         3         4         5         6
##  3.752880  4.441230  7.359933  8.501513 12.690525 14.060153
```

```
identical(predict(aids.1, parameter="mu", type="response"),
          lpred(aids.1, parameter="mu", type="response"))
```

```
## [1] TRUE
```

```
identical(predict(aids.1, type="response"),fitted(aids.1, parameter="mu") )
```

```
## [1] TRUE
```

```
identical(predict(aids.1, type="response"),fv(aids.1, parameter="mu") )
```

```
## [1] TRUE
```

```
identical(predict(aids.1, type="response"),fv(aids.1, parameter="mu") )
```

```
## [1] TRUE
```

`se.fit=TRUE` can be used to obtain approximate standard errors for both `predict()` or `lpred`. The result would be a list containing two objects, `fit` and `se.fit`.

```
paids.1 <- predict(aids.1, what="mu", se.fit=TRUE ,type="response")
names(paids.1)
```

```
## [1] "fit"    "se.fit"
```

```
head(paids.1$se.fit)
```

```
##         1         2         3         4         5         6
## 0.6739890 0.6939025 1.0019629 1.0183976 1.3176894 1.2834913
```

Important: Standard errors should be used with caution. If the (linear) predictor contains only linear (no smoothing) terms then the standard errors of the (linear) predictor (using the option `type="link"`) are correctly calculated. Standard errors for fitted distribution parameters if the link function is not the identity function are calculated using the *delta-method* which could be unreliable, see Chambers and Hastie [1992] p 240. If additive (smoothing) terms are included in the model of a specific distribution parameter then the unreliability increases since the the standard errors of the additive (smoothing) terms are crudely approximated using the method described in Chambers and Hastie [1992] Section 7.4.4.

The option `type="terms"` creates a matrix containing the contribution to the (linear) predictor from each of the terms in the model formula. If in addition the argument `se.fit=TRUE` is set then a list of two objects is created, each containing a matrix. The first matrix contains the contribution of the terms to the (linear) predictor and the second their approximate standard errors. The number of columns in the matrices are the number of terms in the model formula. The argument `terms` can be used in combination with `type="terms"` to select the contribution to the (linear) predictor of a specific term in the model. A typical use of the option `type="terms"` is for plotting the additive contribution of a specific term in modelling a distribution parameter as in the function `term.plot()`.

```
paids.2 <- predict(aids.1, what="mu", type="terms")
colnames(paids.2)

## [1] "poly(x, 3)" "qrt"

# now with se
paids.2 <- predict(aids.1, what="mu", type="terms", se.fit=TRUE)
names(paids.2)

## [1] "fit"    "se.fit"

colnames(paids.2$fit)

## [1] "poly(x, 3)" "qrt"

colnames(paids.2$se.fit)

## [1] "poly(x, 3)" "qrt"

# select only "qrt" to save
paids.2 <- predict(aids.1, what="mu", type="terms", se.fit=TRUE, terms="qrt")
colnames(paids.2$fit)

## [1] "qrt"
```

The most common use of the function `predict()` is to obtain fitted values for a specific parameter at new values of the explanatory variables (predictors) for predictive purposes or for validation. In order to do that the argument `newdata` should be set. The `predict()` function expects that the object given in `newdata` is a data frame containing the right x-variables used in the model. This could cause problems if a transformed variables is used in the fitting of the original model (see below).

The `predict()` function for **gamlss** is based on the `predict.gam()` S-PLUS function of Trevor Hastie which insures that the prediction is reliable even if expressions defining the terms in the model formula depend on the entire data vector for evaluation, are used, see Chambers and Hastie [1992] Section 7.3.3.

We reiterate here the steps used in the execution of `predict()` taken from Chambers and Hastie [1992] Section 7.3.3. Let $\mathbf{D}_{old}$ the original data frame, with original design matrix $\mathbf{X}_{old}$, and $\mathbf{D}_{new}$ the new data frame (the new x-values where the fitted model has to be evaluated) and assume that both data frames contain the right columns in the sense that the x-variables used in the model formula (for the specific distribution parameter) are present in both.

1. Construct a new data frame using the combined (old and new) data, with columns the

matching variables included in both data frames, i.e. $\mathbf{D}_n = \begin{bmatrix} \mathbf{D}_{old} \\ \mathbf{D}_{new} \end{bmatrix}$.

2. Construct the model frame and the corresponding new design matrix, $\mathbf{X}_n = \begin{bmatrix} \mathbf{X}_{old_2} \\ \mathbf{X}_{new} \end{bmatrix}$, using the combined data frame, $\mathbf{D}_n = \begin{bmatrix} \mathbf{D}_{old} \\ \mathbf{D}_{new} \end{bmatrix}$. Note that for certain models (when the function use to construct the design matrix is data dependent) the sub matrix $\mathbf{X}_{old_2}$ of the new design matrix $\mathbf{X}_n$, corresponding to the original observations in $\mathbf{D}_{old}$, may be different from the original design matrix $\mathbf{X}_{old}$. This for example can happen if the cubic spline base, `bs()` is used in the model.
3. The parametric part of the model for the specified parameter is refitted using only $\mathbf{X}_{old_2}$. If the difference of the old and the new fit is large, a warning is given.
4. The coefficients from the fit obtain using only $\mathbf{X}_{old_2}$ are used to obtain the new predictions.
5. If the `gamlss` object contains additive (smoothing) components an additional step is taken to evaluate the appropriate function at the the new data values. (This requires that the additive function has a predict option)

Warning: The `random()`, additive functions does not have a predict option implemented. Predictions for new levels of the factor in `random()` can be obtain by expanding the data to include the new levels and setting the prior weights for the new observations to 1.

Here we use the `aids` data to fit a negative binomial model using a polynomial, `poly()`, a cubic spline base, `bs()`, and a smoothing P-spline, `pb()`, function to model time (x). The `sigma` parameter is a constant is the model. `predict()` is used first, to find values for `mu` (`type="response"`) at new data values and finally for `sigma`. Note that the `predict()` function gives a warning when `bs` is used in the `mu` model.

```
data(aids)
# use with poly
mod1<-gamlss(y~poly(x,3)+qrt, family=NBI, data=aids) #
```

```
## GAMLSS-RS iteration 1: Global Deviance = 383.4541
## . . .
## GAMLSS-RS iteration 4: Global Deviance = 381.7145
```

```
# use with bs
mod2<-gamlss(y~bs(x,5)+qrt, family=NBI, data=aids) #
```

```
# use with pb
 mod3<-gamlss(y~pb(x)+qrt, family=NBI, data=aids)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 373.1785
## . . .
## GAMLSS-RS iteration 5: Global Deviance = 366.9258
```

```
# create a new data frame
newaids<-data.frame(x=c(45,46,47), qrt=c(2,3,4))
# predict "mu" at new values
```

```
(ap1 <- predict(mod1, what="mu", newdata=newaids, type = "response"))
```

```
## Warning in predict.gamlss(mod1, what = "mu", newdata = newaids, type = "response"):
There is a discrepancy  between the original and the re-fit
##  used to achieve 'safe' predictions
##
```

```
## [1] 410.9393 521.6606 471.6455
```

```
(ap2 <- predict(mod2, what="mu", newdata=newaids, type = "response"))
```

```
## Warning in predict.gamlss(mod2, what = "mu", newdata = newaids, type = "response"):
There is a discrepancy  between the original and the re-fit
##  used to achieve 'safe' predictions
##
```

```
## [1] 389.8785 475.1377 408.4420
```

```
(ap3 <- predict(mod3, what="mu", newdata=newaids, type = "response"))
```

```
## new prediction
## [1] 407.0380 513.1502 465.0690
```

```
# get the term contributions
(ap4 <- predict(mod3, what="mu", newdata=newaids, type = "terms"))
```

```
## new prediction
##      pb(x)         qrt
## 1 1.352781 -0.09869534
## 2 1.398161  0.08758643
## 3 1.445596 -0.05823180
## attr(,"constant")
## [1] 4.754821
```

```
(ap4 <- predict(mod3, what="mu", newdata=newaids, type = "terms", se.fit=TRUE))
```

```
## Warning in predict.gamlss(mod3, what = "mu", newdata = newaids, type = "terms",
:  se.fit = TRUE is not supported for new data values at the moment
```

```
## new prediction
##      pb(x)         qrt
## 1 1.352781 -0.09869534
## 2 1.398161  0.08758643
## 3 1.445596 -0.05823180
## attr(,"constant")
## [1] 4.754821
```

```
# predict "sigma"
(ap5 <- predict(mod3, what="sigma", newdata=newaids, type="response"))
```

```
## [1] 0.005131356 0.005131356 0.005131356
```

Note that the `se.fit` argument is not working with new data.

The following example is taken from Venables and Ripley [2002] (who use it to demonstrate that the `predict.lm` function is not working properly for `lm` models). Here we use `gamlss()`

and the safe `predict.gamlss()` function giving consistent correct results.

```
library(MASS)
data(wtloss)
# squaring Days
quad1 <-gamlss(Weight~Days+I(Days^2),data=wtloss)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 137.8867
## GAMLSS-RS iteration 2: Global Deviance = 137.8867
```

```
# using poly
quad2 <-gamlss(Weight~Days+poly(Days,2),data=wtloss)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 137.8867
## GAMLSS-RS iteration 2: Global Deviance = 137.8867
```

```
# new data
new.x <-data.frame(Days=seq(250,300,10), row.names=seq(250,300,10))
# using predict
predict(quad1, newdata=new.x)
```

```
## [1] 112.5061 111.4747 110.5819 109.8277 109.2121 108.7351
```

```
predict(quad2, newdata=new.x)
```

```
## Warning in predict.gamlss(quad2, newdata = new.x): There is a discrepancy between
the original and the re-fit
##  used to achieve 'safe' predictions
##
```

```
## [1] 112.5061 111.4747 110.5819 109.8277 109.2121 108.7351
```

If a transformed variable is used in the fitting of the current data, some care has to taken to insure that the right variables exist in the new data as well. For example, let us assume that a transformation of age is needed in the model i.e. `nage<-age^.5`. This could be fitted as `mod<-gamlss(y ~ cs(age^.5),data=mydata)` or by transforming the age first, `nage<-age^.5`, and then fitting `mod<-gamlss(y~cs(nage), data=mydata)`. The later fit is more efficient particularly for a data set with large number of data cases. In the first case, the code `predict(mod,newdata=data.frame(age=c(34,56)))` would produce the expected results. In the second case a new data frame has to be created containing the old data plus any new transform variable. This data frame has to be declared in the `data` argument of the `predict()` function. The option `newdata` should contain a `data.frame` with the transformed variable names and the transformed variable values for which prediction is required as the following example demonstrates.

```
data(abdom)
# assume that a transformation x^5 is required
aa<-gamlss(y~pb(x^.5),data=abdom, trace=FALSE)
# predict at old values
predict(aa, what="mu")[610]
```

```
## [1] 371.4253
```

```
# predict at new data
predict(aa,newdata=data.frame(x=abdom$x[610]))
```

```
## new prediction
## [1] 371.4253

# now transform x first
nx<-abdom$x^.5
aaa<-gamlss(y~pb(nx),data=abdom, trace=FALSE)
# create a new data frame
newd<-data.frame( abdom, nx=abdom$x^0.5)
# predict at old values
predict(aaa)[610]

## [1] 371.4253

# predict at new values
predict(aaa,newdata=data.frame(nx=abdom$x[610]^.5), data=newd)

## new prediction
## [1] 371.4253
```

5.4 The `gen.likelihood()` function

We have seen in Chapter **??** that the fitting algorithms for GAMLSS models consists of repeatedly calling an iterative weighted least squares algorithm (possible penalised). The whole process is repeated several times until convergence of the the global deviance. The problem with such of procedure is that at the end of the algorithm the standard errors provided by each least squares fitting are not correct because they assume that all the other parameters of the distribution are fixed at their current fitted values. The function `gen.likelihood()` is created to try to overcome this deficiency. Given a fitted `gamlss` model the function `gen.likelihood()` generates the likelihood function of the model so it can be used to create the Hessian matrix required for the construction of the standard errors of the parameters. The function `gen.likelihood()` is used by the `vcov()` function to obtain the right Hessian matrix after a model has fitted. Here is an example on how the function `gen.likelihood()` is working:

```
data(aids)
m100 <- gamlss(y~x+qrt, data=aids, family=NBI, trace=FALSE)
# get the value of log Likelihood
#logLik(m100)
# generate the log likelihood function
logL<-gen.likelihood(m100)
# evaluate it at the final fitted values
logL()

## [1] 246.3187

# the following code is equivalent
logL(c(coef(m100), coef(m100, "sigma")))

## [1] 246.3187

# now getting the Hessian matrix using optimHess()
optimHess(c(coef(m100), coef(m100, "sigma")), logL)
```

```
##             (Intercept)           x         qrt2         qrt3         qrt4
## (Intercept)  212.050893   4971.82091   51.5380919   52.0715125   51.791336
## x           4971.820911 140205.85157 1198.0983500 1187.7321846 1237.550255
## qrt2          51.538092   1198.09835   51.5380919    0.0000000    0.000000
## qrt3          52.071512   1187.73218    0.0000000   52.0715125    0.000000
## qrt4          51.791336   1237.55026    0.0000000    0.0000000   51.791336
## (Intercept)    1.826129    -15.03447    0.6425665    0.2461583    0.231789
##             (Intercept)
## (Intercept)   1.8261293
## x           -15.0344720
## qrt2          0.6425665
## qrt3          0.2461583
## qrt4          0.2317890
## (Intercept)  18.1635333
```

When smoothing terms are fitted the function `gen.likelihood()` considers them as fixed at their fitted value so the Hessian in this case does not take into account the variability for the fitting of the smoothers.

```
m200 <- gamlss(y~pb(x)+qrt, data=aids, family=NBI, trace=FALSE)
# create the log Likelihood
logL2<-gen.likelihood(m200)
# evaluate it at the final fitted values
logL2(c(coef(m200), coef(m200, "sigma")))
```

```
## [1] 183.4629
```

```
# now getting the Hessian matrix
optimHess(c(coef(m200), coef(m200, "sigma")), logL2)
```

```
##              (Intercept)         pb(x)         qrt2         qrt3
## (Intercept) 3.741335e+03  1.111319e+05   848.198810   947.019761
## pb(x)       1.111319e+05  3.695284e+06 24596.919051 27506.266553
## qrt2        8.481988e+02  2.459692e+04   848.198810     0.000000
## qrt3        9.470198e+02  2.750627e+04     0.000000   947.019761
## qrt4        9.279800e+02  2.811189e+04     0.000000     0.000000
## (Intercept) 1.183395e+00 -3.405392e+00     3.595562    -7.024548
##                     qrt4 (Intercept)
## (Intercept)   927.980000    1.183395
## pb(x)       28111.889104   -3.405392
## qrt2            0.000000    3.595562
## qrt3            0.000000   -7.024548
## qrt4          927.980000    2.294323
## (Intercept)     2.294323    5.416412
```

The entry under the `pb(x)` column refer to the linear part of the smoother and therefore should not be interpreted on their own but only in combination with the fitted smoother.

5.5 The vcov() and rvcov() functions

The generic function `vcov()` within the package **gamlss** uses the function `gen.likelihood()` to construct numerically the Hessian matrix. Standard errors for the estimated coefficients are obtained from the observed information matrix (that is, the inverse of the Hessian). The standard errors obtained this way are more reliable, that the ones obtained during the fitting GAMLSS algorithms since they take into account the information about the interrelationship between the distribution parameters, i.e. μ, σ, ν and τ. The function `rvcov()` creates the robust or sandwich standard errors. Here are the arguments of the generic function `vcov()`.

This function is not visible

```
vcov.gamlss(object, type = c("vcov", "cor", "se", "coef", "all"),
                    robust = FALSE, hessian.fun = c("R", "PB"), ...)
```

where

`object` a `gamlss` fitted object,

`type` what is required i) variance-covariance matrix, `"vcov"`, ii) the correlation matrix, `"cor"`, iii) the standard errors, `"se"`, iv) the fitted coefficients, `"coef`, v) all the above as a list, `"all"`.

`robust` whether the normal (FALSE) or robust or sandwich (TRUE) standard errors are required

`hessian.fun` how to obtain numerically the Hessian i) using `optimHess()`, option `"R"` ii) using a function by Pinheiro and Bates taken from package `nlme`, option `"PB"`.

`...` for extra arguments

As an example we use the model fitted in the previous section:

```
# the correlation betwwen the fitted parameters
vcov(m100, type="cor")
```

```
##             (Intercept)            x         qrt2        qrt3         qrt4
## (Intercept)  1.00000000 -0.760891874 -0.462032219 -0.47515350 -0.446719862
## x           -0.76089187  1.000000000  0.018879191  0.03463720 -0.002258329
## qrt2        -0.46203222  0.018879191  1.000000000  0.47791869  0.476834481
## qrt3        -0.47515350  0.034637199  0.477918686  1.00000000  0.477952149
## qrt4        -0.44671986 -0.002258329  0.476834481  0.47795215  1.000000000
## (Intercept) -0.08157924  0.088524725  0.001656598  0.01247367  0.009506204
##              (Intercept)
## (Intercept) -0.081579245
## x            0.088524725
## qrt2         0.001656598
## qrt3         0.012473668
## qrt4         0.009506204
## (Intercept)  1.000000000
```

```
# the standard errors
vcov(m100, type="se")
```

```
## (Intercept)           x        qrt2        qrt3        qrt4 (Intercept)
## 0.204805619 0.006535937 0.192532079 0.192104635 0.192260793 0.235686941

# the sandwich standard errors
vcov(m100, type="se", robust=TRUE)

## (Intercept)           x        qrt2        qrt3        qrt4 (Intercept)
## 0.291009501 0.008784397 0.212083065 0.210080980 0.198288097 0.250504613
```

The function `rvcov()` has the same arguments as the function `vcov.gamlss()` apart from the argument `robust`. It provides the sandwich or robust standard errors. Robust standard errors, introduced by Huber [1967] and White [1980] are, in general, more reliable than the usual standard errors when the variance model is suspected not to be correct (assuming the mean model is correct). The sandwich standard errors are usually (but not always) bigger than the usual ones. Here is an example of simulated data from a gamma distribution with `sigma=2`, where an incorrect exponential distribution model is fitted instead. Therefore we would expect the robust standard errors to be greater that the standard ones and more reliable.

```
#set seed
set.seed(4321)
# gererate from a gamma distribution
Y <- rGA(200, mu=1, sigma=2)
# fitting the wrong model i.e. sigma=1
r1 <- gamlss(Y~1, family=EXP)

## GAMLSS-RS iteration 1: Global Deviance = 391.2369
## GAMLSS-RS iteration 2: Global Deviance = 391.2369

# the conventional se is too precise
vcov(r1, type="se")

## (Intercept)
##  0.07071067

# the sandwich se is wider
rvcov(r1, type="se")

## (Intercept)
##   0.1182156

# fitting the correct model
 r2 <- gamlss(Y~1, family=GA)

## GAMLSS-RS iteration 1: Global Deviance = 19.5225
## GAMLSS-RS iteration 2: Global Deviance = 19.5225

 # standard se's
 vcov(r2, type="se")

## (Intercept) (Intercept)
##  0.13170785  0.03935216

  # robust se's
 rvcov(r2, type="se")
```

```
## (Intercept) (Intercept)
##  0.11851375  0.03866188

# similar stadard errors
# also obtained using
vcov(r2, type="se", robust=TRUE)

## (Intercept) (Intercept)
##  0.11851375  0.03866188
```

5.6 The summary() and confint() functions

More detailed information about the fitted GAMLSS model than the method `print()` , which provides only limited information, is obtained using the generic function `summary.gamlss()`. The arguments of the function are as follows:

```
summary(object, type = c("vcov", "qr"),
                      robust=FALSE, save = FALSE,
                      hessian.fun = c("R", "PB"), ...)
```

where

`object`	a `gamlss` fitted object,
`type`	the default value `"vcov"` uses the `vcov()` method for gamlss to get the variance-covariance matrix of the estimated beta coefficients. The alternative `"qr"` produces standard errors from the individual least square fits but it is not reliable since it does not take into the account the inter-correlation between the distributional parameters μ, σ, ν and τ.
`robust`	whether the robust (or sandwich) standard errors are required in which case the function `rvcov()` is called.
`save`	whether to save the environment of the function in order to have access to its values
`hessian.fun`	how to obtain numerically the Hessian. Using the `optimHess()` function, option `"R"` or using the Pinheiro and Bates function taken from package `nlme`, option `"PB"`
`...`	for extra arguments

In the following example the environment of the function `summary()` is saved so say the p-values can be accessed.

```
sm100 <-summary(m100, robust=TRUE, save=TRUE)

## ******************************************************************
## Family:  c("NBI", "Negative Binomial type I")
##
## Call:
## gamlss(formula = y ~ x + qrt, family = NBI, data = aids, trace = FALSE)
##
```

```
## Fitting method: RS()
##
## ------------------------------------------------------------------
## Mu link function:  log
## Mu Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.885458   0.291010   9.915 3.25e-12 ***
## x            0.087433   0.008784   9.953 2.92e-12 ***
## qrt2        -0.120383   0.212083  -0.568    0.574
## qrt3         0.111753   0.210081   0.532    0.598
## qrt4        -0.075539   0.198288  -0.381    0.705
## ---
## 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)  -1.6032     0.2505    -6.4 1.44e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## ------------------------------------------------------------------
## No. of observations in the fit:  45
## Degrees of Freedom for the fit:  6
##       Residual Deg. of Freedom:  39
##                       at cycle:  3
##
## Global Deviance:     492.6373
##             AIC:     504.6373
##             SBC:     515.4773
## ******************************************************************
```

```
names(sm100)
```

```
##  [1] "ps"          "co"          "p1"          "pm"          "est.disp"
##  [6] "coef.table"  "pvalue"      "tvalue"      "se"          "coef"
## [11] "ifWarning"   "covmat"      "object"      "type"        "robust"
## [16] "save"        "hessian.fun" "digits"
```

```
sm100$pvalue
```

```
##  (Intercept)            x         qrt2         qrt3         qrt4
## 3.253778e-12 2.921666e-12 5.735457e-01 5.977766e-01 7.053023e-01
##  (Intercept)
## 1.443488e-07
```

For testing the significance of individual terms given all the rest of the terms are in the model it may be better to use the `drop1()` function instead of relying on p-values from `summary()`. The `drop1()` function provides the generalised likelihood ratio test (GLRT) for dropping each

term which is much more reliable than the Wald test based on the standard errors given by the p-value above. The GLRT has an asymptotic Chi-squared distribution with degrees of freedom equal to the number of parameters in the term dropped. This only applies if the model does not include smoothing term(s). In the present of smoothing terms in the model, `drop1()` could be used as rough guide to the significance of each of the parametric terms, provided the smoothing degrees of freedom are fixed at theirs values chosen from the model prior to `drop1()`. Note that for complicated models with large data `drop1()` can take few minutes. Here we first apply `drop1()` to a fully parametric model `m100`. The resulting test p-values 0.661 for the parametric term `qrt` should be reliable assuming thr parametric submodel (`x`) was correct.

```
drop1(m100)
```

```
## Single term deletions for
## mu
##
## Model:
## y ~ x + qrt
##        Df    AIC    LRT Pr(Chi)
## <none>    504.64
## x       1 576.91 74.271  <2e-16 ***
## qrt     3 500.23  1.593   0.661
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Here in model `m300` we fix the smoothing degrees of freedom (from model `m200` which includes a smoothing term) and then apply `drop1(m300)`.

```
m300 <- gamlss(y~pb(x, df=m200$mu.nl.df)+qrt, data=aids, family=NBI, trace=FALSE)
drop1(m300)
```

```
## Single term deletions for
## mu
##
## Model:
## y ~ pb(x, df = m200$mu.nl.df) + qrt
##                                Df    AIC     LRT   Pr(Chi)
## <none>                            390.10
## pb(x, df = m200$mu.nl.df) 6.5889 576.91 199.983 < 2.2e-16 ***
## qrt                       3.0000 402.49  18.389 0.0003656 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The resulting test p-value is 3.7×10^{-4} for the parametric term `qrt`. This gives a guide to testing `qrt`. The corresponding test not fixing the smoothing degrees of freedom is certainly not reliable.

The generic function `confint()` provides (standard error based) confidence intervals for the the fitted coefficients. It has the following arguments:

```
confint.gamlss((object, parm, level = 0.95,
                        what = c("mu", "sigma", "nu", "tau"),
                        parameter = NULL, robust = FALSE, ...) )
```

where

`object` a `gamlss` fitted object,

`parm` which term are to be given confidence intervals, either a vector of numbers or a vector of names. If missing, all terms are considered.

`level` the confidence level required.

`what` or `parameter` which distribution parameter to consider.

`robust` whether the normal (FALSE) or robust or sandwich (TRUE) standard errors should be used

`...` for extra arguments

Here is an example using `confint()`:

```
confint(m100)
```

```
##                    2.5 %    97.5 %
## (Intercept)  2.48404626 3.2868695
## x             0.07462303 0.1002434
## qrt2         -0.49773874 0.2569731
## qrt3         -0.26476513 0.4882712
## qrt4         -0.45236355 0.3012849
```

```
confint(m100,1, robust=TRUE)
```

```
##                2.5 %   97.5 %
## (Intercept) 2.31509 3.455826
```

5.7 The `prof.dev()` and `prof.term()` functions

There are two function providing profile likelihood intervals for a GAMLSS model.

`prof.dev()`: which can be used to obtain profile deviance plot for any *parameters* μ, σ, ν or τ of the distribution.

`prof.term()`: which can be used to obtain profile deviance plot for any linear *term* in a model for the distribution parameters μ, σ, ν or τ.

5.7.1 `prof.dev()`

The function `prof.dev()` obtains a profile deviance plot for any of the distribution parameters μ, σ, ν or τ of the fitted family and is useful for checking the reliability of models in which one (or more) of the parameters in the distributions are constant, (and therefore have not been modelled as functions of explanatory variables). The `prof.dev()` also provides a 100(1-α)% profile likelihood confidence interval for the parameter (which is, in general, much more reliable than a standard error based confidence interval for a parameter) for a specified value of α.

```
prof.dev(object, which = NULL, min = NULL, max = NULL,
        step = NULL, length = 7, startlastfit = TRUE,
        plot = TRUE, perc = 95, ...)
```

where

`object`	a `gamlss` fitted object,
`which`	which parameter to get the profile deviance e.g. `which=``tau''`
`min`	the minimum value for the parameter e.g. `min=1`
	`max` the maximum value for the parameter e.g. `max=20`
`step`	how often to evaluate the global deviance (defines the step length of the grid for the parameter) e.g. `step=1`
`length`	if `step` is not set, this gives how many times the function has to be evaluate for the construction of the profile deviance, the default value is equal 7.
`startlastfit`	whether to start fitting from the last fit or not, default value is `startlastfit=TRUE`.
`plot`	whether to plot, `plot=TRUE` or save the results, `plot=FALSE`
`perc`	what % confidence interval is required
...	for extra arguments

As an example consider the abdominal circumference model fitted in Section 5.2 where a t distribution is fitted to to the data with smooth terms for the μ and σ but not for ν which is fitted as a constant. The ν parameter is the degrees of freedom parameter of the t distribution and it would be of some interest to find a confidence interval for it. Note that $\nu = 1$ corresponds to a Cauchy distribution while a large value of ν corresponds closely to a normal distribution. Usually it takes several attempts to select a suitable range for the parameter in order to produce a decent graph. As a default (if the argument `step` is not specified) the profile deviance is evaluated at only 7 points and a cubic spline approximation of the function is formed. This can produce a wobbly function. The arguments `step` or `length` can be then used to improve the approximation. Our advice is to start with a sparse grid for the parameter (i.e. few points) and improve that when you see the resulting plot (aiming to include the full 95% confidence interval for the parameter within the horizontal axis scale and the horizontal deviance bar representing the global deviance at the endpoints of the parameter interval to be roughly half way up the vertical axis scale). Note that the procedure requires fitting the model repeatedly for a sequence of fixed values of the parameter of interest (ν in this example) so it can be slow.

Here we reproduce our first attempt (shown at the left side of Figure 5.1) and our final attempt (shown at the right side of Figure 5.1).

```
pd1<-prof.dev(h,"nu",min=5, max=50)
```

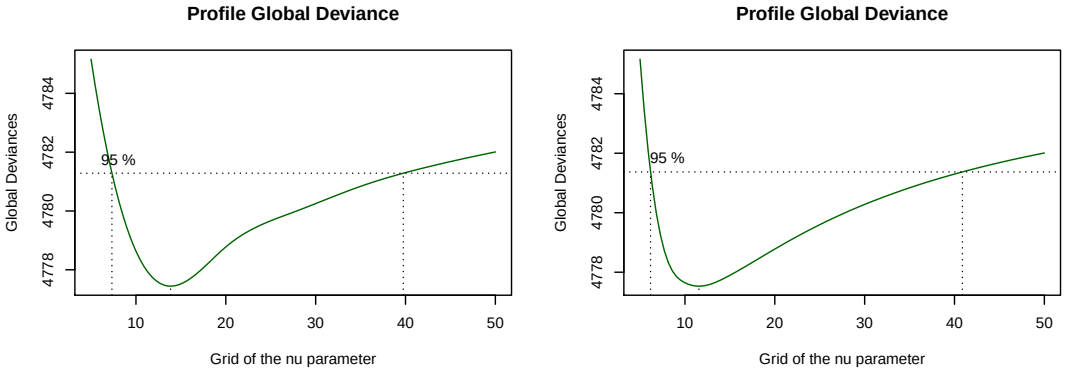
```
## ********************************************************************
## . . .
## ********************************************************************
## The Maximum Likelihood estimator is  13.84947
## with a Global Deviance equal to  4777.443
## A  95 % Confidence interval is: ( 7.324271 , 39.75037 )
```

```
## ********************************************************************
```

Now we increase the evaluation of the function to 20.

```
pd2<-prof.dev(h,"nu",min=5, max=50, length=20)
```

```
## ********************************************************************
## . . .
## ********************************************************************
## The Maximum Likelihood estimator is  11.55543
## with a Global Deviance equal to  4777.53
## A  95 % Confidence interval is: ( 6.171182 , 40.87693 )
## ********************************************************************
```



R code on page 118

Figure 5.1: Profile deviance for ν from a t-family fitted model `h` using `abdom` data with $\mu = pb(x)$ and $\log(\sigma) = pb()/$ The left panel has 7 evaluation of the function while the right panel has 20.

Note that the object `pd2` has several components saved, among them the profile deviance function under the name `fun`. This function can be used for further evaluations of he function as the following code shows. Beware of not trying to evaluated outside the original range since this can be very misleading:

```
names(pd2)
```

```
## [1] "values"    "fun"       "min"       "max"       "max.value" "CI"
## [7] "criterion"
```

```
pd2$fun(34)
```

```
## [1] 4780.734
```

```
curve(pd2$fun(x), 5, 50)
```

R code on page 119

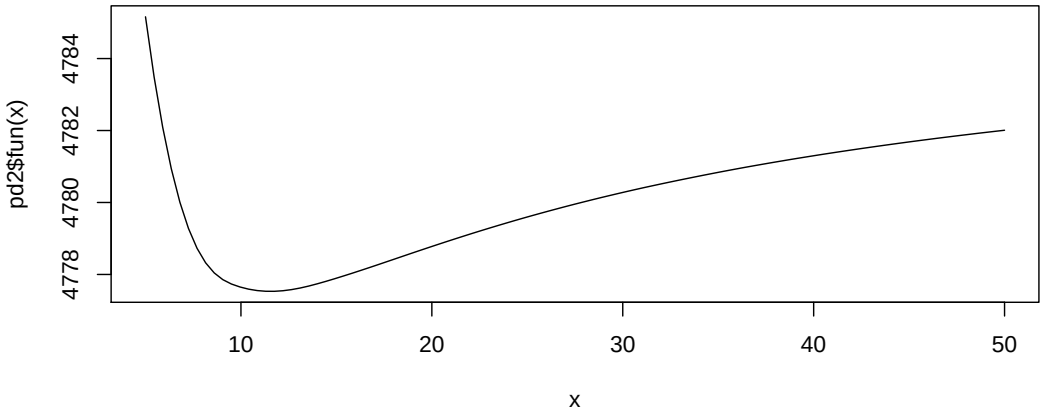


Figure 5.2: The profile deviance for ν plotted using `curve()`.

For different confidence intervals change the `perc` option, e.g. for a 99% use `perc=99`.

5.7.2 `prof.term()`

The function `prof.term()` is similar to the function `prof.dev()` but it can provide a profile deviance for any parameter in the model not just for the distribution parameters. That is, while `prof.dev()` can be applied to profile a (constant) parameter of the distribution of the response variable y (i.e. μ, σ, ν or τ), the `prof.term()` can be applied to any parameter in the predictor model for μ, σ, ν or τ. In order to show how the `prof.term()` is working consider the `aids` data used in Section 5.4. Let us assume first that we are interested to fit a linear in time term (x) plus a factor for the the quarterly seasonal effect, `qrt`, using the negative binomial model (type I) family. This model is fitted as `gamlss(y~x + qrt, family = NBI, data = aids)`.

The coefficient for the linear term in time (x) has the value of 0.08743 and a t-value of 13.3773 which indicates that it is highly significant from zero. An approximate 95% confidence interval for this parameter can be obtained using the function `confint()`:

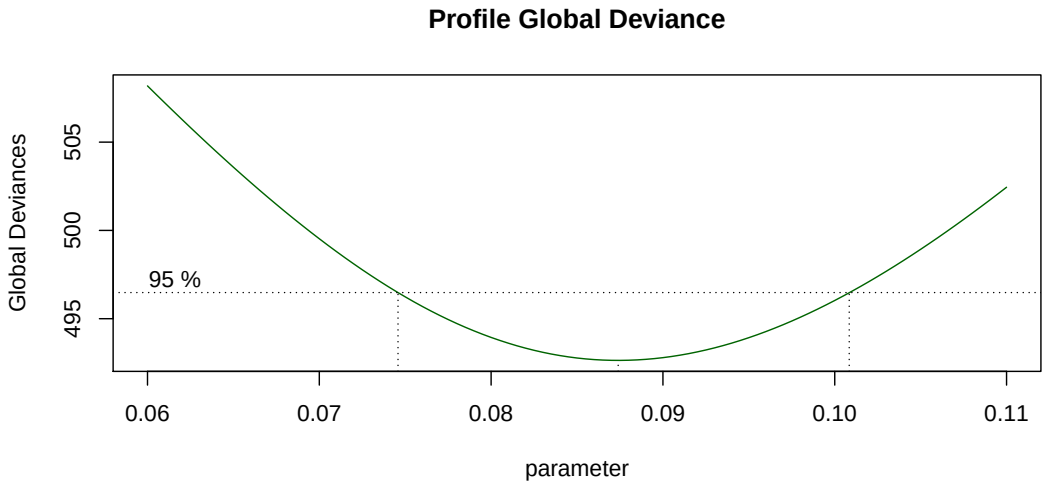
```
confint(m100, "x")
```

```
##        2.5 %    97.5 %
## x 0.07462303 0.1002434
```

We shall use now the function `prof.term` to find hopefully a more accurate profile (deviance) 95% confidence interval for the linear term parameter. Note that `this` in the model formula indicates which parameter to profile.

```
mod<-quote(gamlss(y ~ offset(this * x) + qrt, data = aids, family = NBI))
prof.term(mod, min=0.06, max=0.11)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 508.2
## GAMLSS-RS iteration 2: Global Deviance = 508.2
## GAMLSS-RS iteration 3: Global Deviance = 508.2
## GAMLSS-RS iteration 1: Global Deviance = 500.8
## . . .
## ********************************************************************
## The Maximum Likelihood estimator is  0.08739
## with a Global Deviance equal to  492.6
## A  95 % Confidence interval is: ( 0.07458 , 0.1008 )
## ********************************************************************
```



R code on page 120

Figure 5.3: The profile deviance for the coefficint of `x`.

The profile deviance looks quadratic so it is not a surprise that the approximate 95% confidence interval and the 95% profile interval are almost identical. In general this would not be the case if the likelihood is not nearly quadratic at the maximum. To obtain a 99% interval use `prof.term(mod, min=0.06, max=0.11, perc=99)`.

Now we shall used `plot.term()` to find a break point in the relationship between the response and one of the explanatory variables. Stasinopoulos and Rigby [1992] have shown that the AIDS data provide a clear break point between the AIDS cases and time. Here we consider a linear+linear model for time (x), i.e. `x+(x>break)*(x-break)` and we are interested to estimate the break point., Stasinopoulos and Rigby [1992].

```
aids.1 <- quote(gamlss(y ~ x+I((x>this)*(x-this))+qrt,family=NBI,data=aids))
prof.term(aids.1, min=15, max=23, length=15,  criterion="GD")
```

```
## GAMLSS-RS iteration 1: Global Deviance = 403.8474
## GAMLSS-RS iteration 2: Global Deviance = 402.9138
## GAMLSS-RS iteration 3: Global Deviance = 402.913
## GAMLSS-RS iteration 1: Global Deviance = 396.6076
```

```
## . . .
## *********************************************************************
## The Maximum Likelihood estimator is  18.33295
## with a Global Deviance equal to  377.4614
## A  95 % Confidence interval is: ( 17.19218 , 19.41849 )
## *********************************************************************
```

R code on page 121

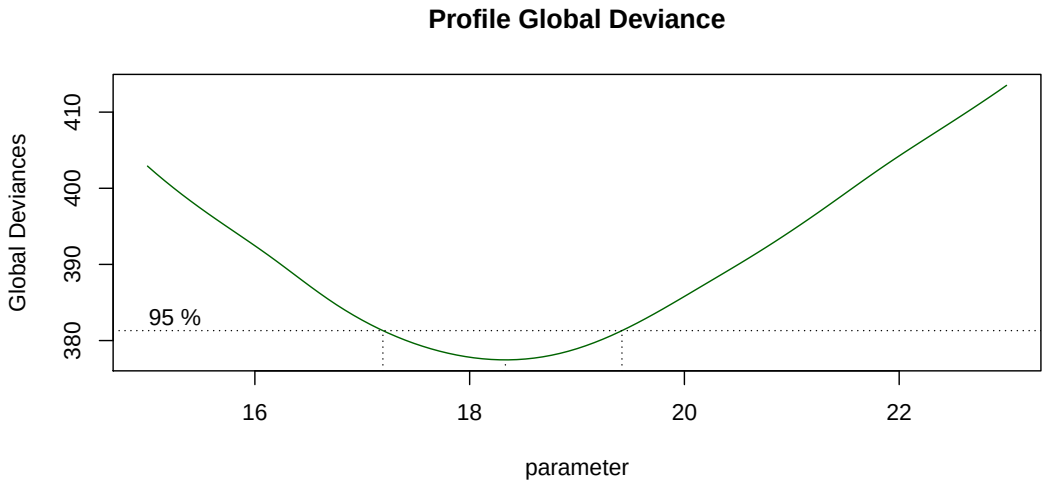


Figure 5.4: The profile deviance for the break point parameter of `x`.

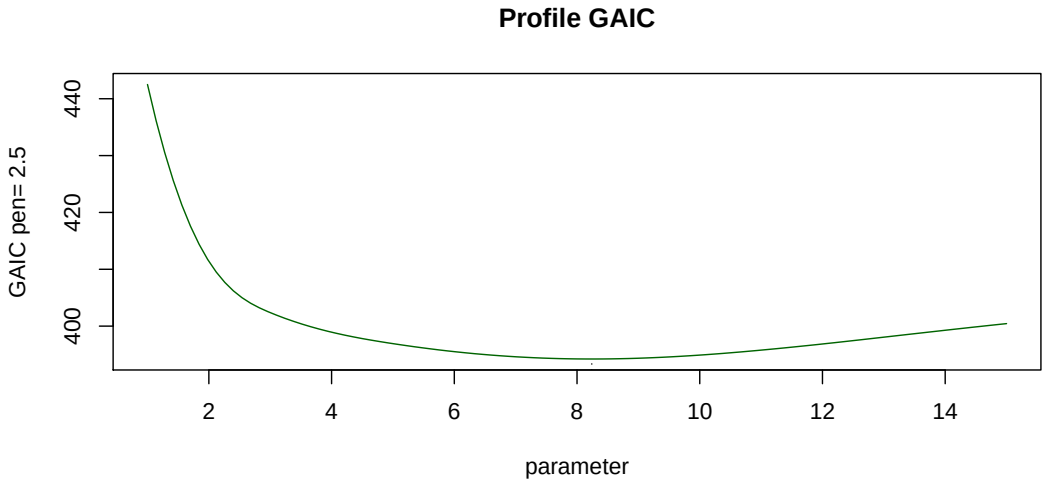
The profile plot shown in Figure (5.4) suggests strong support for a break point.

Finally the function `prof.term()` can also be used as a way of determining a smoothing (hyper) parameter in a model by plotting the Generalized Akaike Information Criterion, GAIC(k) [where penalty k is specified by the `penalty` argument of `prof.term`]. Consider the model `gamlss(y ~ cs(x,df=??) + qrt, data = aids, family = NBI)` in which we would like to determine a reasonable value for the missing degrees of freedom of the cubic spline function `cs()`. (Note that, in `pb()` the smoothing parameter and therefore the degrees of freedom are estimated automatically using a local maximum likelihood procedure, while here the estimation of effective degrees of freedom is done globally). Models with different degrees of freedom can be fitted and their generalized Akaike Information criterion (GAIC) plotted against the degrees of freedom. This process can be automated using the function `prof.term()`.

```
mod1<-quote(gamlss(y ~ cs(x,df=this) + qrt, data = aids, family = NBI))
prof.term(mod1, min=1, max=15, step=1, criterion="GAIC", penalty=2.5)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 419.651
## GAMLSS-RS iteration 2: Global Deviance = 423.8293
## GAMLSS-RS iteration 3: Global Deviance = 425.0032
## GAMLSS-RS iteration 4: Global Deviance = 425.0032
## . . .
## GAMLSS-RS iteration 3: Global Deviance = 347.9341
```

```
## ********************************************************************
## The Mimimum is  8.240494
## with an an GAIC( 2.5 ) = 394.2163
## ********************************************************************
```



R code on page 122

Figure 5.5: Profile GAIC with penalty 2.5 for the degrees of freedom in the model `gamlss(y cs(x,df=this) + qrt, data = aids, family = NBI)`.

The profile GAIC plot, with penalty= 2.5, shown in Figure 5.5 suggests support for effective degrees of freedom around 8. Note that GAIC with penalty 2.5 denoted $GAIC(2.5) = -2\log(\hat{\ell}) + 2.5 * df$, where $\hat{\ell}$ is the fitted likelihood for a given values of the degrees of freedom parameter df of the cubic splines smoother. Alternative penalties values could be used e.g.. penalty= $2, 3, 4$ or $\log(n)$.

Important: Profile deviance intervals should be used with care if random effects are included in the model for any of the distribution parameters. They correspond to a naive plug-in profile estimation which in general produces narrower intervals than the marginal likelihood approach, see Rigby and Stasinopoulos [2005] Section 6.2 and Appendix A.2. The more accurate profile deviance intervals are obtained from the approximate marginal likelihoods which are model dependent. At present we do not provide a general function for calculating these intervals but see comments below.

Finding a confidence interval for a parametric term parameters when the model contains smoothing terms is more difficult. One possible approach which could provide a guide to the confidence internals is as follows:

i) fit the full model including smoothing terms on which the smoothing parameter or the effective degrees of freedom could be estimated i.e. `pb(x)`

ii) fix each of the smoothing degrees of freedom to their values from the full model in i)

iii) profile the parametric term parameters in model ii)

Part III

Distributions

Chapter 6

The `gamlss.family` of distributions

This chapter:

1. describe the different types of distribution within `gamlss`
2. how to visualise the different distributions
3. how to create a new distribution and
4. about link functions within `gamlss`

This chapter is essential for understanding the different types of distributions in GAMLSS and especially the need for more complex distributions.

6.1 Introduction

Within the GAMLSS framework the population probability (density) function of the response variable Y, $f(y|\boldsymbol{\theta})$, where $\boldsymbol{\theta} = (\mu, \sigma, \nu, \tau)$, is deliberately left general with no explicit distribution specified. The only restriction that the **R** implementation of GAMLSS, Stasinopoulos and Rigby (2007), has for specifying the distribution of Y is that the function $f(y|\boldsymbol{\theta})$ and its first (and optionally expected second and cross) derivatives with respect to each of the parameters of $\boldsymbol{\theta}$ must be computable. Explicit derivatives are preferable, but numerical derivatives can be used (resulting in reduced computational speed). That is, the algorithm used for fitting the regression model needs only this information.

Here we introduce the available distributions within the current implementation of GAMLSS in **R**. We refer to this set of distributions as the GAMLSS family to be consistent with **R** where the distributions are defined as `gamlss.family` objects. Note that comprehensive review of all `gamlss.family` distributions can be found in the book "Distributionf for Location Scale and Shape".

Fitting a parametric distribution within the GAMLSS family can be achieved using the command `gamlss(`$y \sim 1$`, family="" )` where the argument `family` can take any `gamlss.family` distribution, see Tables 6.1, 6.2 and 6.3 for appropriate names. For example, in order to fit say a negative binomial distribution to some count data one can use `family=NBI`. Note also the following forms are acceptable: `family=NBI()`, `family="NBI"` or `family=NBI(mu.link=log, sigma.link=log)` with the `NBI` default link functions for μ and σ which can be amended. Here is an example of fitting a distribution to the Turkish stock exchange returns data shown in Figure 6.1:

```
data(tse)
truehist(tse$ret, xlab="TSE",main="")
m1 <- gamlss(ret~1, data=tse, family=TF)
```

```
## GAMLSS-RS iteration 1: Global Deviance = -12734.83
## . . .
## GAMLSS-RS iteration 11: Global Deviance = -12795.56
```

R code on page 128

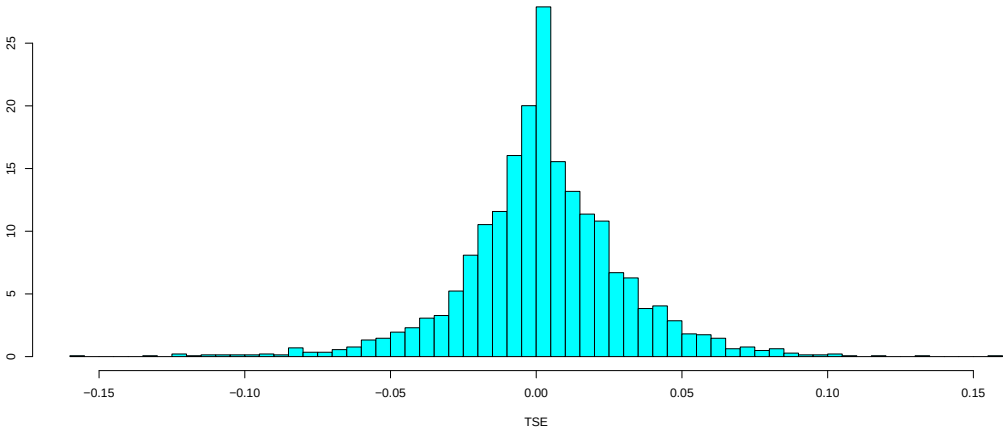


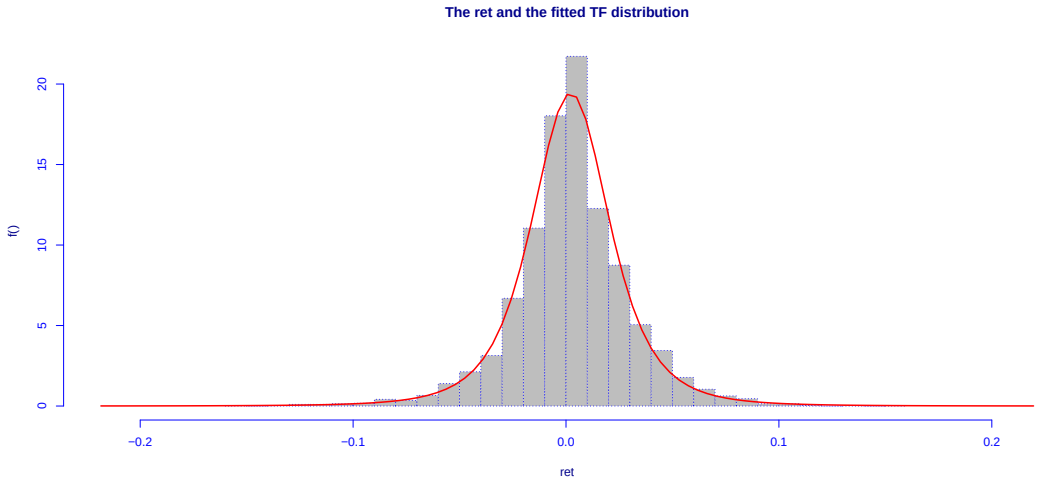
Figure 6.1: A histogram of the Turkish stock exchange returns.

When no explanatory x-variables are involved, (as above), the functions `gamlssML()`, `histDist()` and `fitDist()` can be used instead of `gamlss()`. `gamlssML()` uses optimization techniques in **R** and it is faster then the algorithms that `gamlss()` uses, which are designed for regression type models.

```
m2 <- gamlssML(ret, data=tse, family=TF)
```

`histDist()` uses `gamlssML()` as a default algorithm to fit the model but in addition displays the histogram together with the fitted distribution of the data, see Figure 6.2,

```
m3 <- histDist(ret, data=tse, family=TF, nbins=30)
```



R code on page 128

Figure 6.2: A histogram of the Turkish stock exchange returns together with a fitted t distribution.

The function `fitDist()` uses `gamlssML()` to fit a set of predetermine distribution to the data and chooses the "best" according to a Generalised Akaike Information criterion (GAIC). The order of the fitted models can be displayed as shown below:

```
m5 <- fitDist(ret, data=tse, type="realline")
m5$fits
```

```
##      SEP2      SEP1      SEP3      SEP4        PE        GT    SHASHo
## -12879.01 -12876.88 -12876.14 -12865.04 -12862.09 -12860.09 -12836.07
##       JSU       ST3        TF       ST2       ST5       ST1       ST4
## -12803.68 -12790.80 -12789.56 -12788.83 -12788.20 -12787.75 -12787.62
##        LO        NO       SN2       SN1        GU        RG
## -12726.41 -12444.12 -12442.92 -12442.12 -11578.60 -11305.11
```

The book "Distributions for LOcatiob Scale and Shape" contains more examples demonstrating all of the above those functions.

6.2 Types of distribution within the GAMLSS family

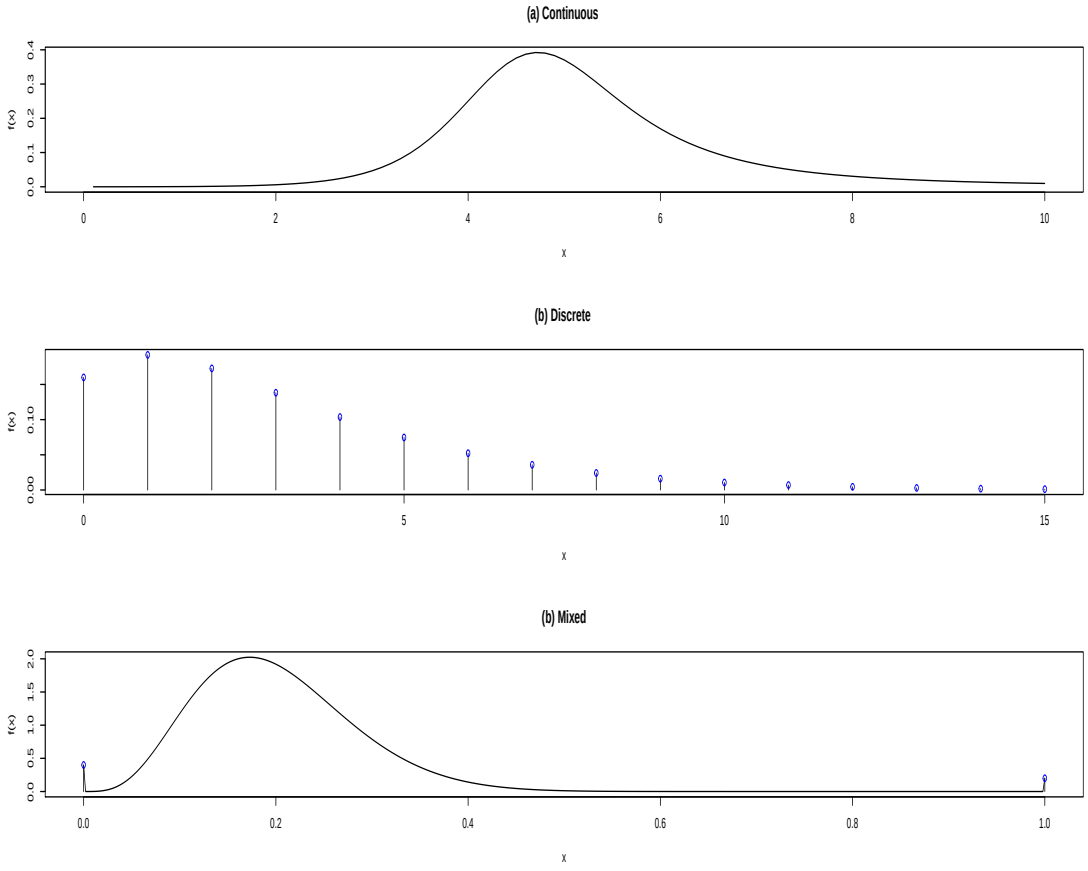
6.2.1 Explicit GAMLSS family distributions

The type of distribution to use depends on the type of the response variable. Within the GAMLSS family there are three distinct types of distributions:

1. continuous distributions see Figure 6.3(a),
2. discrete distributions, see Figure 6.3(b),
3. mixed distributions see Figure 6.3(c).

Figure 6.3

```
op<-par(mfrow=c(3,1))
curve(dBCT(x, mu=5, sigma=0.2, nu=-1, tau=3), 0,10, xlab="x",
      ylab="f(x)", main="(a) Continuous ")
plot(x<-0:15, dNBI(x, mu=3, sigma=.5), type="h", xlab="x",
     ylab="f(x)", main="(b) Discrete ")
points(x, dNBI(x, mu=3, sigma=.5), col="blue")
plotBEINF( mu =.2 , sigma=.2, nu = 1, tau = 0.5, from = 0, to=1,
           n = 501, ylab="f(x)",  main="(b) Mixed")
par(op)
```



R code on page 130

Figure 6.3: Different type of distributions in GAMLSS (s) continuous, (b) discrete, (c) mixed

All the continuous distributions in `gamlss.family` are shown in Table 6.1. The columns of the Table shows the names, the `gamlss.family` names and the default link functions of the distribution respectively. Link functions were introduced by Weddeburn and Nelder (1972) for Generalised Linear Models but are appropriate for all regression models since they guarantee that the distribution parameters remain within the appropriate range. For example take the

beta distribution in the first row of Table 6.1. Both μ and σ are defined on $(0, 1)$. The `logit` link for μ uses the predictor $\eta = \log(\frac{\mu}{1-\mu})$ for fitting the μ parameter in the model. Therefore $\mu = \frac{e^{\eta}}{1+e^{\eta}}$ which ensures that μ is in the right range from 0 to 1.

The continuous distributions, $f_Y(y|\boldsymbol{\theta})$ in Table 6.1 are defined on $(-\infty, +\infty)$, $(0, +\infty)$ and $(0, 1)$, ranges. Users can restrict those ranges of the response variable Y by defining a truncated `gamlss.family` distribution using the package `gamlss.tr`.

Discrete distributions $P(Y = y|\boldsymbol{\theta})$ are usually defined on $y = 0, 1, 2, \ldots, n$, where n is a known finite value or n is infinite, i.e. usually discrete (count) values. Table 6.2 shows the available discrete `gamlss.family` distributions.

Mixed distributions are a special case of finite mixture distributions described in Chapter **??** and are mixtures of continuous and discrete distributions, i.e. continuous distributions where the range of Y has been expanded to include some discrete values with non-zero probabilities. These distributions are useful for modelling data like insurance claims where most of the people are not claiming therefore there is a high probability at zero, but if they claim then the distribution on the amount of claim is defined in the positive line. The zero inflated gamma distribution shown in Figure **??** is a possible distribution for such data. Table 6.3 shows the available mixed `gamlss.family` distributions.

Distributions	R Name	$\boldsymbol{\mu}$	$\boldsymbol{\sigma}$	$\boldsymbol{\nu}$	$\boldsymbol{\tau}$
beta	`BE()`	`logit`	`logit`	-	-
Box-Cox Cole-Green	`BCCG()`	`identity`	`log`	`identity`	-
Box-Cox power exp.	`BCPE()`	`identity`	`log`	`identity`	`log`
Box-Cox t	`BCT()`	`identity`	`log`	`identity`	`log`
exponential	`EXP()`	`log`	-	-	-
exponential Gaussian	`exGAUS()`	`identity`	`log`	`log`	-
exponential gen. beta t2	`EGB2()`	`identity`	`identity`	`log`	`log`
gamma	`GA()`	`log`	`log`	-	-
generalised beta type 1	`GB1()`	`logit`	`logit`	`log`	`log`
generalised beta type 2	`GB2()`	`log`	`identity`	`log`	`log`
generalised gamma	`GG()`	`log`	`log`	`identity`	-
generalised inv. Gaussian	`GIG()`	`log`	`log`	`identity`	-
generalised t	`GT()`	`identity`	`log`	`log`	`log`
Gumbel	`GU()`	`identity`	`log`	-	-
inverse Gamma	`IGAMMA()`	`log`	`log`	-	-
inverse Gaussian	`IG()`	`log`	`log`	-	-
Johnson's SU repar.	`JSU()`	`identity`	`log`	`identity`	`log`
Johnson's original SU	`JSUo()`	`identity`	`log`	`identity`	`log`
logistic	`LO()`	`identity`	`log`	-	-
logit normal	`LOGITNO()`	`logit`	`log`	-	-
log normal	`LOGNO()`	`identity`	`log`	-	-
log normal 2	`LOGNO2()`	`log`	`log`	-	-
log normal (Box-Cox)	`LNO()`	`identity`	`log`	fixed	-
NET	`NET()`	`identity`	`log`	fixed	fixed
normal	`NO()`	`identity`	`log`	-	-
normal family	`NOF()`	`identity`	`log`	`identity`	-
Pareto 2 original	`PARETO2o()`	`log`	`log`	-	-

Pareto 2	PARETO2()	log	log	-	-
Pareto 2 repar	GP()	log	log	-	-
power exponential	PE()	identity	log	log	-
reverse Gumbel	RG()	identity	log	-	-
skew normal type 1	SN1()	identity	log	identity	-
skew normal type 2	SN2()	identity	log	identity	-
skew power exp. t1	SEP1()	identity	log	identity	log
skew power exp. t2	SEP2()	identity	log	identity	log
skew power exp. t3	SEP3()	identity	log	log	log
skew power exp. t4	SEP4()	identity	log	log	log
sinh-arcsinh original	SHASHo()	identity	log	identity	log
sinh-arcsinh original 2	SHASHo2()	identity	log	identity	log
sinh-arcsinh	SHASH()	identity	log	log	log
skew t type 1	ST1()	identity	log	identity	log
skew t type 2	ST2()	identity	log	identity	log
skew t type 3	ST3()	identity	log	log	log
skew t type 3 repar	SST()	identity	log	log	logshifto2
skew t type 4	ST4()	identity	log	log	log
skew t type 5	ST5()	identity	log	identity	log
t Family	TF()	identity	log	log	-
t Family repar	TF()	identity	log	logshifto2	-
Weibull	WEI()	log	log	-	-
Weibull (PH)	WEI2()	log	log	-	-
Weibull (μ the mean)	WEI3()	log	log	-	-

Table 6.1: Continuous distributions implemented within the gamlss.dist package(with default link functions)

For the **R** implementation of GAMLSS all of the distributions in Tables 6.1 and 6.2 have d, p, q and r functions corresponding respectively to the probability (density) function (pdf), the cumulative distribution function (cdf), the quantiles (i.e. inverse cdf) and random value generating functions. For example, the gamma distribution has the functions dGA, pGA, qGA and rGA. In addition each distribution has a fitting function which helps the fitting procedure by providing link functions, first and (exact or approximate) expected second derivatives, starting values etc. All fitting functions have as arguments the link functions for the distribution parameters. For example, the fitting function for the gamma distribution is called GA with arguments mu.link and sigma.link. The default link functions for all gamlss.family distributions are shown in columns 3-6 of Tables 6.1, 6.2 and 6.3. The function show.link() can be used to identify which are the available links for the distribution parameter within each of the gamlss.family. For example,

```
show.link(BCT)

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

Distributions	R Name	μ	σ	ν
beta binomial	BB()	logit	log	-
binomial	BI()	logit	-	-
geometric	GEOM()	log	-	-
logarithmic	LG()	logit	-	-
Delaporte	DEL()	log	log	logit
negative binomial type I	NBI()	log	log	-
negative binomial type II	NBII()	log	log	-
Poisson	PO()	log	-	-
Poisson inverse Gaussian	PIG()	log	log	-
Sichel	SI()	log	log	identity
Sichel (μ the mean)	SICHEL()	log	log	identity
Waring (μ the mean)	WARING()	log	log	-
Yule (μ the mean)	YULE()	log	-	-
zero altered beta binomial	ZABB()	logit	log	logit
zero altered binomial	ZABI()	logit	logit	-
zero altered logarithmic	ZALG()	logit	logit	-
zero altered neg. binomial	ZANBI()	log	log	logit
zero altered poisson	ZAP()	log	logit	-
zero inflated beta binomial	ZIBB()	logit	log	logit
zero inflated binomial	ZIBI()	logit	logit	-
zero inflated neg. binomial	ZINBI()	log	log	logit
zero inflated poisson	ZIP()	log	logit	-
zero inflated poisson (μ the mean)	ZIP2()	log	logit	-
zero inflated poisson inv. Gaussian	ZIPIG()	log	log	logit

Table 6.2: Discrete distributions implemented within the **gamlss** packages (with default link functions)

beta inflated (at 0)	BEOI()	logit	log	logit	-
beta inflated (at 0)	BEINF0()	logit	logit	log	-
beta inflated (at 1)	BEZI()	logit	log	logit	-
beta inflated (at 1)	BEINF1()	logit	logit	log	-
beta inflated (at 0 and 1)	BEINF()	logit	logit	log	log
zero adjusted GA	ZAGA()	log	log	logit	-
zero adjusted IG	ZAIG()	log	log	logit	-

Table 6.3: Mixed distributions implemented within the **gamlss** packages (with default link functions)

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

will display the available links within the `BCT` distribution. Available link functions are the usual `glm()` link functions plus some extra like `logshiftto1`, and `own`, see Section 6.4. The `own` option allows the user to define his/her own link function, for an example see the help file on the function `make.link.gamlss()` e.g. `?make.link.gamlss`.

6.2.2 Extending GAMLSS family distributions

There are several ways to extend the `gamlss.family` distributions. This can be achieved by

- creating a new `gamlss.family` distribution,
- creating a *log* or *logit* version of a distributions from an existing continuous `gamlss.family` distribution on the real line $(-\infty, \infty)$,
- truncating an existing `gamlss.family`,
- using a censored version of an existing `gamlss.family`
- mixing different `gamlss.family` distributions to create a new finite mixture distribution.

New `gamlss.family` distributions

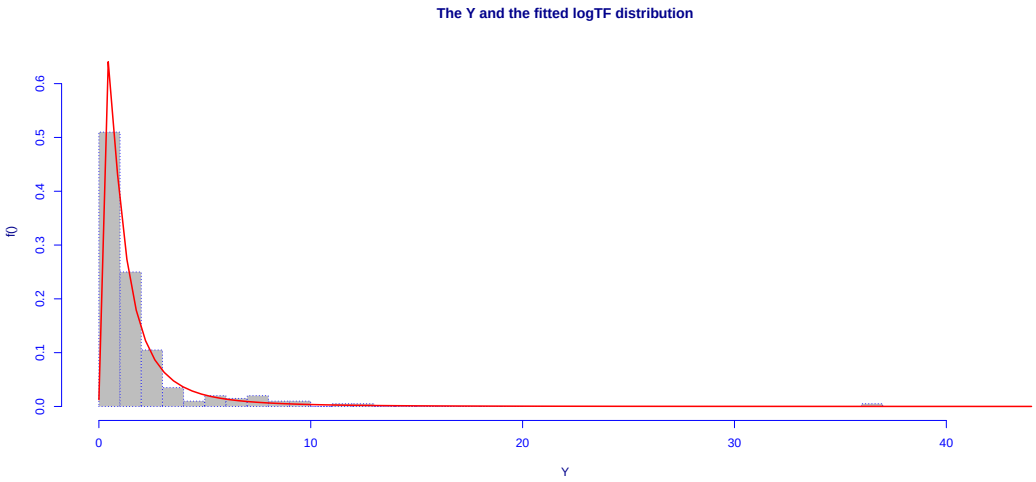
To create a new `gamlss.family` distribution is relatively simple, if the pdf function of the distribution can be evaluated easily. To do that, find a file of a current `gamlss.family` distribution, (having the same number of distribution parameters) and amend accordingly. Section 6.4 provides an example on how to do that.

New *log* and *logit* versions from a continuous `gamlss.family` on $(-\infty, \infty)$

Any random variable say Z defined on continuous distribution in $(-\infty, +\infty)$ can be transformed using $Y = \exp(Z)$ to a random variable defined on the positive scale $(0, \infty)$. The typical example of this is the log-normal distribution which is defined by $Y = \exp(Z)$ where Z is a normally random variable. The function `gen.Familily()` using the option `type="log"` can do that. Here is an example in which we create a log-t distribution on $(0, \infty)$, generate a random sample of 200 observations from the distribution and finally fit the distribution to the generated data.

```
# generate the distribution
gen.Family("TF", type="log")
```

```
## A  log  family of distributions from TF has been generated
##  and saved under the names:
##  dlogTF plogTF qlogTF rlogTF logTF

# generate 200 observations
set.seed(345)
Y<- rlogTF(200)
# fit the distribution
h1 <- histDist(Y, family=logTF, nbins=30, ylim=c(0,.65))
```



R code on page 135

Figure 6.4: A fitted log-t to 200 simulated observations

Similarly a logit-t distribution on $(0, 1)$ can be created using the following code:

```
gen.Family("TF", type="logit")

## A  logit  family of distributions from TF has been generated
##  and saved under the names:
##  dlogitTF plogitTF qlogitTF rlogitTF logitTF
```

Truncating `gamlss.family` distributions

Truncating existing `gamlss.family` distributions can be achieved by using the add-on package **gamlss.tr**. The function `gen.trun()`, within the **gamlss.tr** package, can take any `gamlss.family` distribution and generate the `d`, `p`, `q`, `r` and `R` fitting functions for the specified truncated distribution. The truncation can be left, right or in both tails of the range of the response y variable.

```
library(gamlss.tr)
gen.trun(par=c(0,100),family="TF", name="0to100", type="both")

## A truncated family of distributions from TF has been generated
```

```
##  and saved under the names:
##  dTF0to100 pTF0to100 qTF0to100 rTF0to100 TF0to100
## The type of truncation is both and the truncation parameter is 0 100

set.seed(123)
Y<-rTF0to100(1000, mu=80 ,sigma=20, nu=5)
h1 <- histDist(Y, family=TF0to100, nbins=30, xlim=c(0,100))
```

R code on page 136

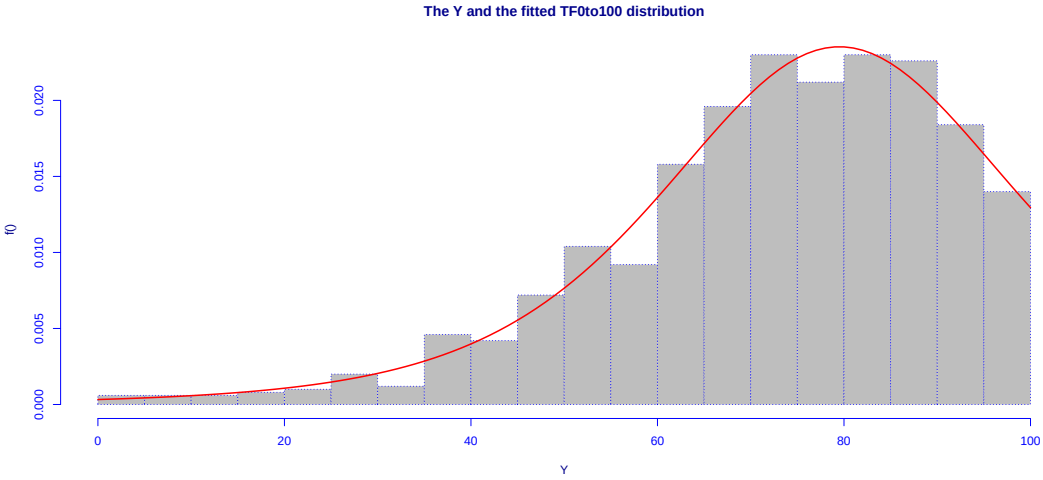


Figure 6.5: A fitted truncated t distribution defined on $0, 100$, fitted to simulated 1000 observations

Note that for continuous distributions left truncation at 3 means that the random variable can take the value 3. For discrete distributions left truncation at 3 means that the random variable can take values from 4 onwards. Also for discrete distributions right truncation at 10 means that the random variable can take values up to 10.

Censored `gamlss.family` distributions

The package **gamlss.cens** is designed for the situation where the response variable is left or right censored or, more generally, it has been observed in an interval form, e.g.. $(3, 10]$ an interval from 3 to 10 (including only the right end point 10). The function `gen.cens()` will take any `gamlss.family` distribution and create a new function which can fit a response of "interval" type. Note that for "interval" response variables the usual likelihood function for independent response variables defined as

$$L(\boldsymbol{\theta}) = \prod_{i=1}^{n} f(y_i|\boldsymbol{\theta}) \tag{6.1}$$

changes to

$$L(\boldsymbol{\theta}) = \prod_{i=1}^{n} [F(y_{2i}|\boldsymbol{\theta}) - F(y_{1i}|\boldsymbol{\theta})] \tag{6.2}$$

where $F(y)$ is the cumulative distribution function and $(y_{1i}, y_{2i}]$ is the observed interval. The following is an example of generating a Weibull distribution which allows an "interval" response variable to be fitted. The data are called `lip` and come from a experimental enzymology research project which attempted to develop a generic food spoilage model. Note that the response variable `lip$y` is defined as an interval response.

```
library(gamlss.cens)
data(lip)
head(lip$y, 10)
```

```
##  [1]  1-       1-       1-       1-      [11, 18]  1-       1-
##  [8]  1-       1-      [ 2,  4]
```

The value $1-$ indicates an interval $(1, \infty)$ not including 1, while $[11, 18]$ indicates the interval $(11, 18]$ not including 11 but including 18. For a continuous distribution the likelihood is unaffected by whether interval endpoints are included or not, but for a discrete distribution this is very important.

```
gen.cens(WEI2,type="interval")
```

```
## A censored family of distributions from WEI2 has been generated
##  and saved under the names:
##  dWEI2ic pWEI2ic qWEI2ic WEI2ic
## The type of censoring is interval
```

```
WEI2ic
```

```
##
## GAMLSS Family: WEI2ic interval censored Weibull type 2
## Link function for mu    : log
## Link function for sigma: log
```

```
weimi<- gamlss(y ~ poly(Tem,2)+poly(pH,2)+poly(aw,2), data=lip,
     family=WEI2ic, c.crit=0.00001, n.cyc=200, trace=FALSE)
summary(weimi)
```

```
## ******************************************************************
## Family:  c("WEI2ic", "interval censored Weibull type 2")
##
## Call:  gamlss(formula = y ~ poly(Tem, 2) + poly(pH, 2) + poly(aw, 2),
##     family = WEI2ic, data = lip, c.crit = 1e-05, n.cyc = 200,
##     trace = FALSE)
##
## Fitting method: RS()
##
## ------------------------------------------------------------------
## Mu link function:  log
## Mu Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -5.6495     0.7591  -7.443 2.17e-11 ***
## poly(Tem, 2)1  37.0182     4.7789   7.746 4.62e-12 ***
## poly(Tem, 2)2  -2.1235     2.0769  -1.022   0.3088
## poly(pH, 2)1   22.0409     3.3806   6.520 2.10e-09 ***
```

```
## poly(pH, 2)2   -4.6537     2.1145  -2.201   0.0298 *
## poly(aw, 2)1   33.0681     4.5549   7.260 5.45e-11 ***
## poly(aw, 2)2    1.6015     1.9023   0.842   0.4017
## ---
## 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)   0.1813     0.1550   1.169    0.245
##
## ------------------------------------------------------------------
## No. of observations in the fit:  120
## Degrees of Freedom for the fit:  8
##       Residual Deg. of Freedom:  112
##                       at cycle:  200
##
## Global Deviance:     138.146
##             AIC:     154.146
##             SBC:     176.4459
## ******************************************************************
```

Use `WEI2` for a proportional hazard model or `WEI3` for μ = population mean.

Finite mixtures of `gamlss.family` distributions

Finite mixtures of `gamlss.family` distributions can be fitted using the package **gamlss.mx**. A finite mixture of `gamlss.family` distributions will have the form

$$f_Y(y|\boldsymbol{\psi}) = \sum_{k=1}^{K} \pi_k f_k(y|\boldsymbol{\theta}_k) \tag{6.3}$$

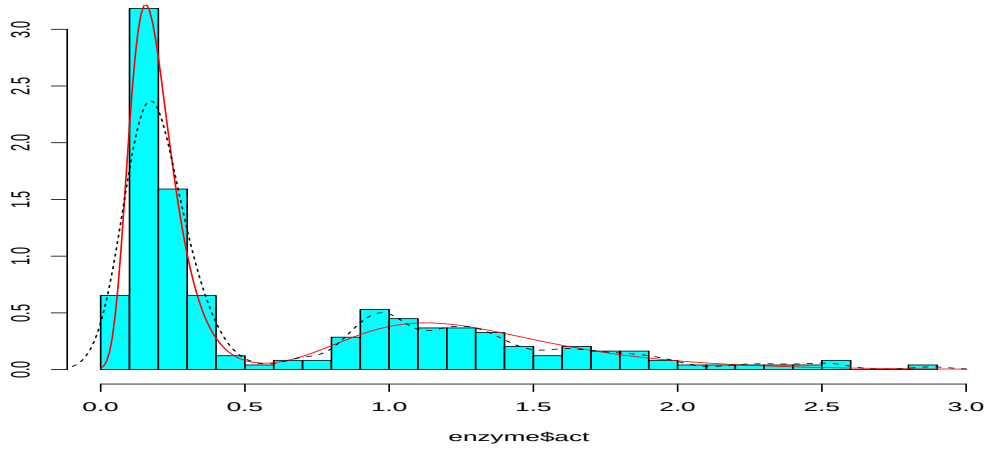
where $f_k(y|\boldsymbol{\theta}_k)$ is the probability (density) function of y for component k, and $0 \leq \pi_k \leq 1$ is the prior (or mixing) probability of component k, for $k = 1, 2, \ldots, K$. Also $\sum_{k=1}^{K} \pi_k = 1$ and $\boldsymbol{\psi} = (\boldsymbol{\theta}, \boldsymbol{\pi})$ where $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_k)$ and $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_K)$. Any combination of (continuous or discrete) `gamlss.family` distributions can be used. The model in this case is fitted using the EM algorithm. The component probability (density) functions may have different parameters [fitted using the function `gamlssMX()`] or may have parameters in common [fitted using the function `gamlssNP()`]. In the former case, the mixing probabilities may also be modelled using explanatory variables and the finite mixture may have a zero component (e.g. zero inflated negative binomial etc.). Both functions `gamlssMX()` and `gamlssNP()` are in the add on package **gamlss.mx**. Chapter **??** gives more details about modelling and fitting finite mixtures models using the package **gamlss.mx**. Figure **??** shows an example of fitting a finite mixture of two reverse Gumbel distributions to the `enzyme` data.

```
library(gamlss.mx)
data(enzyme)
```

```
m3 <- gamlssMX(act ~ 1, data = enzyme, family = RG, K = 2)
```

```
library(MASS); library(gamlss.mx)
truehist(enzyme$act, h = 0.1)
 fyRG <- dMX(y = seq(0, 3, 0.01), mu = list(1.127, 0.1557),
             sigma = list(exp(-1.091), exp(-2.641)),
                pi = list(0.376, 0.624), family = list("RG","RG"))
lines(seq(0, 3, 0.01), fyRG, col = "red", lty = 1)
lines(density(enzyme$act, width = "SJ-dpi"), lty = 2)
```

Figure 6.6



R code on page 139

Figure 6.6: Showing a fitted reverse Gumbel finite mixture with two components distribution to the *enzyme* data (continuous line) together with fitted non-parametric density estimate (dash line)

6.3 Displaying GAMLSS family distributions

Each GAMLSS family distribution has five functions. The "fitting" function which is used in the argument `family` of the `gamlss()` function when fitting a distribution and the usual four **R** functions, `d`,`p`, `q` and `r` for the pdf, the cdf, the inverse cdf and the random generating function respectively. The names of the fitting `gamlss.family` functions are given in column two of Tables 6.1 6.2 and 6.3 respectively

For example the pdf, cdf, inverse cdf and random generating functions of the normal distribution who has within the `gamlss.family` the name `NO` are given as `dNO`, `pNO`, `qNO`, `rNO` respectively.

6.3.1 Using the distribution demos

A `gamlss.family` population distribution can be displayed graphically in **R** using the `gamlss.demo` package. For example the following commands will bring the **gamlss.demo** package and start the gamlss demos.

```
library(gamlss.demo)
gamlss.demo()
```

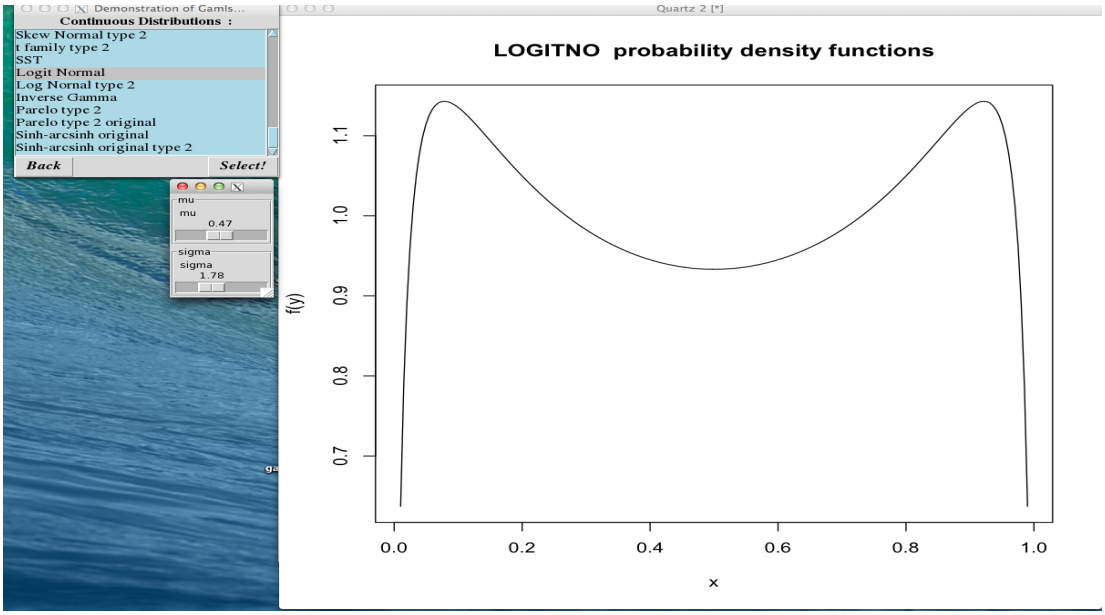


Figure 6.7: Showing a screen shot demonstrating the logit Normal distribution, `LOGITNO`

This will display a menu where by choosing the option "gamlss family distributions" you can proceed to display the different distributions. Alternatively you can just type `demo.NAME()` where `NAME` is a `gamlss.family` name e.g. `demo.NO()` for normal distribution. This allows any distribution in GAMLSS to be displayed graphically and its parameters adjusted interactively. A screen shot the how this looks like is given in Figure 6.7.

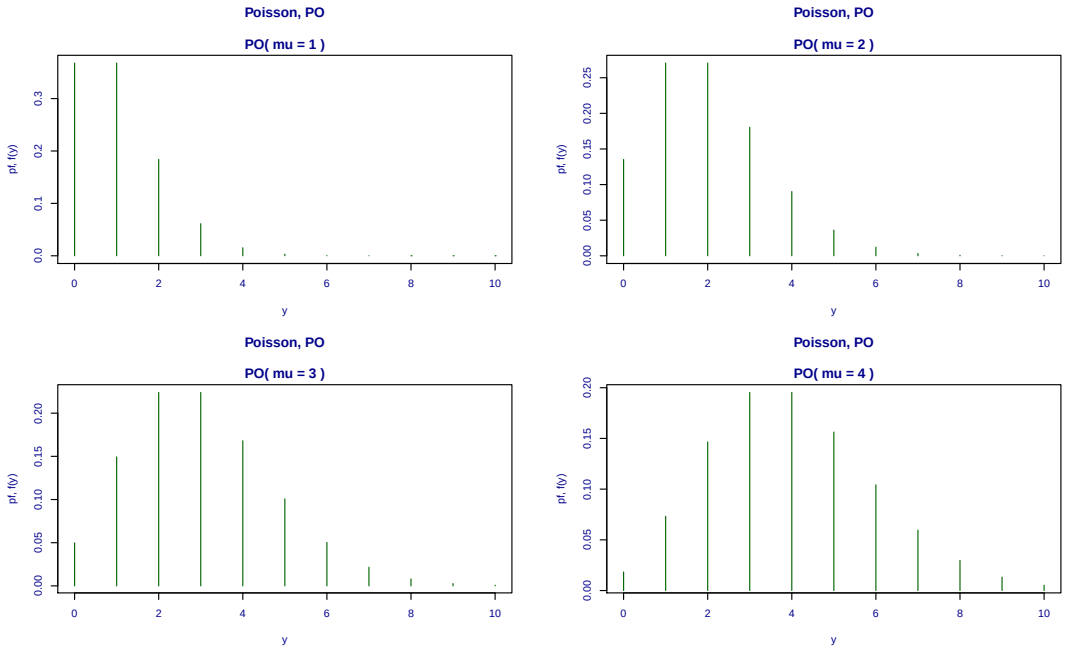
6.3.2 Using the `pdf.plot()` function

An alternative method of graphically displaying the probability (density) functions is to use the `pdf.plot()` function:

Figure 6.8

```
pdf.plot(family=PO(),mu=c(1,2,3,4), min=0, max=10,step=1)
```

The resulting figure is shown in Figure 6.8.



R code on page 140

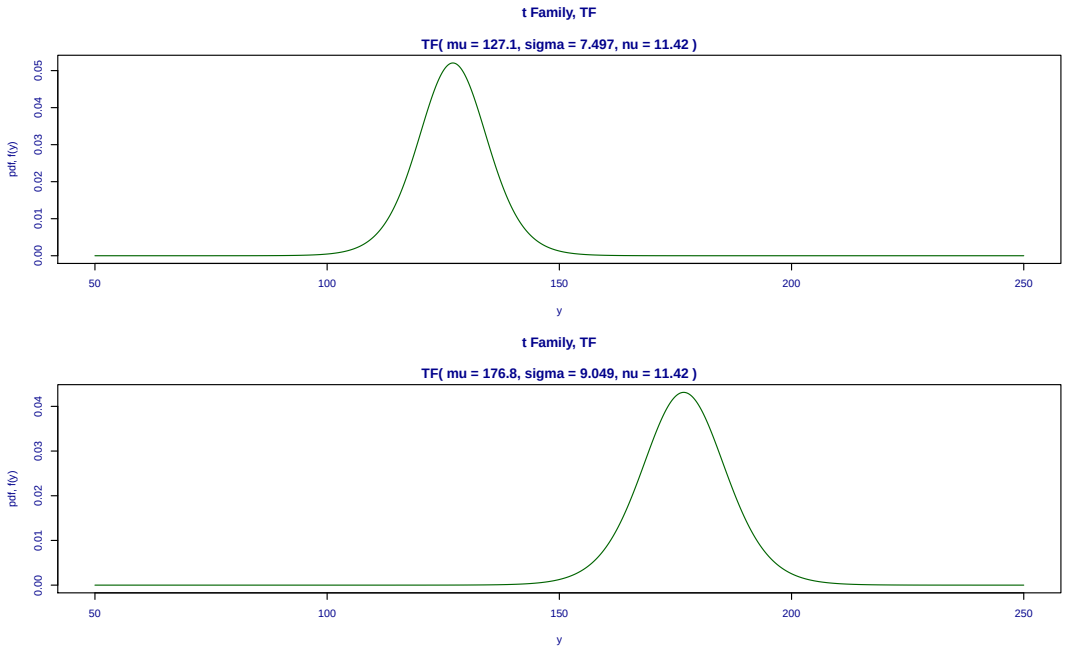
Figure 6.8: Plotting the Poison distribution using the `pdf.plot()` function

This function is also useful for plotting different fitted distributions for specific observations. For example here we plot the fitted distribution for observations 100 and 200 after we have fitted a t-distribution to the `abdom` data.

Figure 6.9

```
m1 <- gamlss(y~pb(x), sigma.fo=~pb(x), data=abdom, family=TF, trace=FALSE)
pdf.plot(m1,obs=c(100,200), min=50, max=250,step=.1)
```

```
pdf.plot(m1,obs=c(100,200), min=50, max=250,step=.1)
```



R code on page 141

Figure 6.9: Plotting the fitted distribution for observations 100 and 200

The resulting figure is shown in Figure 6.9.

6.3.3 Plotting the `d`, `p`, `q` and `r` functions of a distribution

The following code plot demonstrate how to plot the `d`, `p`, `q` and `r` functions of a continuous distribution.

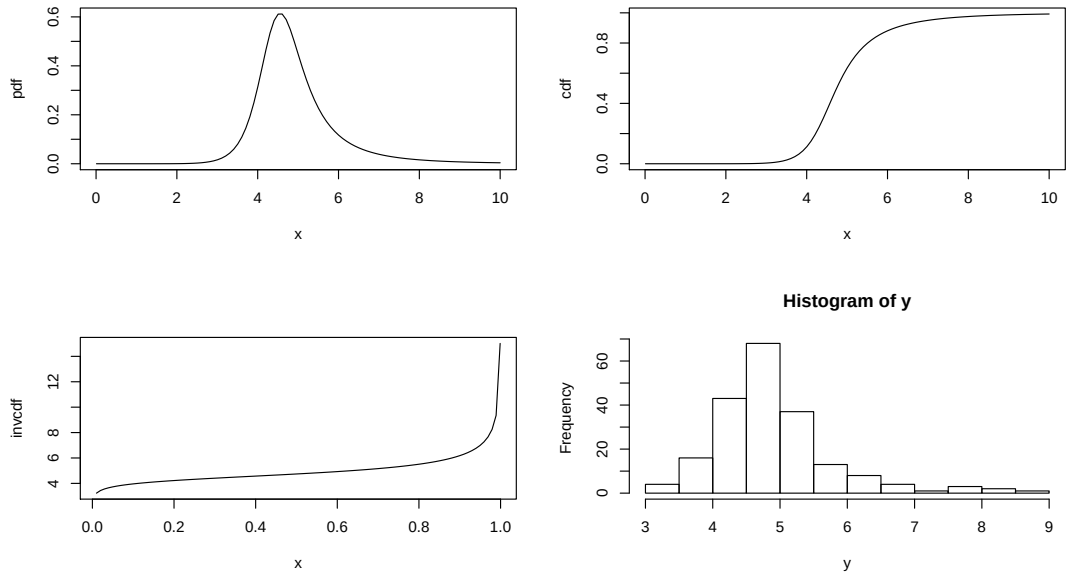
Figure 6.10

```
PPP <- par(mfrow=c(2,2))
   curve(dBCT(x, mu=5, sigma=.2, nu=-5, tau=2), 0.01, 10, ylab="pdf") # pdf
   curve(pBCT(x,  mu=5, sigma=.2, nu=-5, tau=2), 0.01, 10, ylab="cdf")  # cdf
 plot(function(x)  qBCT(x,  mu=5, sigma=.2, nu=-5, tau=2), 0.01, .999, ,
            ylab="invcdf") # inverse cdf
 y<-rBCT(200,  mu=5, sigma=.2, nu=-5, tau=2 ) # randomly generated values
 hist(y)
 par(PPP)
```

The plot appears in Figure 6.10. For discrete distribution use

Figure 6.11

```
PPP <- par(mfrow=c(2,2))
   curve(dBCT(x, mu=5, sigma=.2, nu=-5, tau=2), 0.01, 10, ylab="pdf") # pdf
   curve(pBCT(x,  mu=5, sigma=.2, nu=-5, tau=2), 0.01, 10, ylab="cdf")  # cdf
 plot(function(x)  qBCT(x,  mu=5, sigma=.2, nu=-5, tau=2), 0.01, .999, ,
            ylab="invcdf") # inverse cdf
```



R code on page 142

Figure 6.10: Plotting the `d`, `p`, `q` and `r` functions of a continuous distribution

```
y<-rBCT(200,  mu=5, sigma=.2, nu=-5, tau=2 ) # randomly generated values
hist(y)
par(PPP)
```

The plot is shown in Figure 6.11.

6.4 Amending and constructing a new distribution

Note: This Section can be omitted if the user does not plan to add a new distribution or amend an existing distribution.

This section describes the structure of a `gamlss.family` distribution and how it can be amended to produce a new distribution. As we have mention above for each new distribution five different functions are required. Taking the normal distribution as an example, we have:

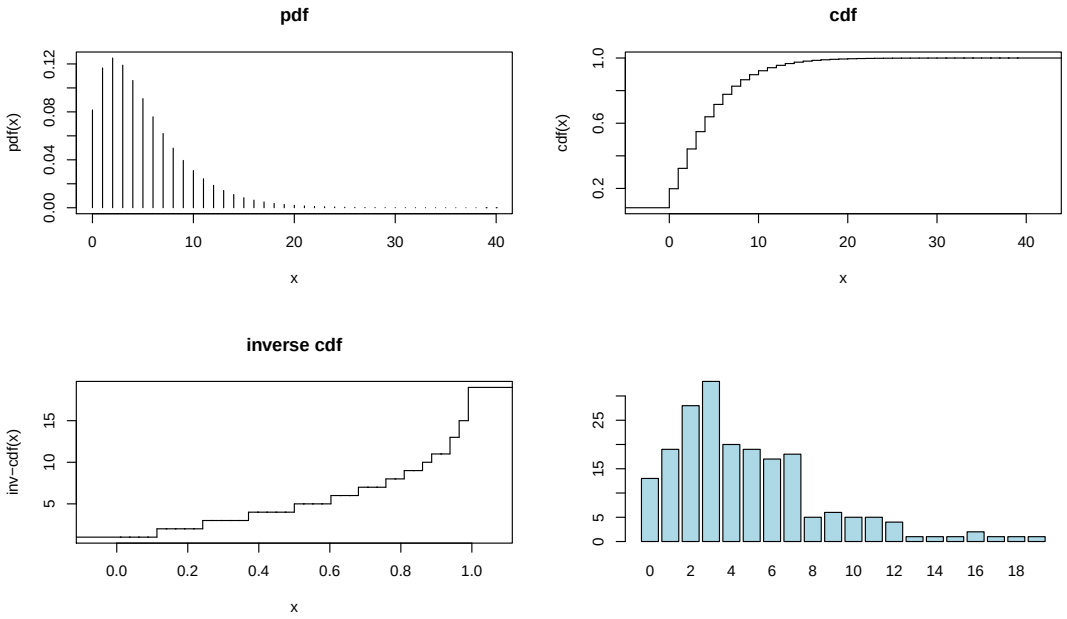
`NO` the function used for fitting

`dNO` the pdf function

`pNO` the cdf function

`qNO` the inverse cdf function

`rNO` the randomization function



R code on page 142

Figure 6.11: Plotting the `d`, `p`, `q` and `r` functions of a discrete distribution

The function, `NO` provides information for fitting the normal distribution within **gamlss**. The function body for `NO()` has three fields:

- the definition of the link functions
- the information needed for fitting the distribution and
- the class definition of the fitted object

Here is how `NO()` is implemented:

```
NO <- function (mu.link = "identity", sigma.link = "log")
{
   mmstats <- checklink("mu.link", "Normal", substitute(mu.link),
        c("inverse", "log", "identity", "own"))
    dstats <- checklink("sigma.link", "Normal", substitute(sigma.link),
        c("inverse", "log", "identity", "own"))
```

The code shows that the available links for both `mu` and `sigma` parameters are the `identity`, `log` and `inverse` links but the default links are `identity` and `log` respectively.

The definition of the link functions

To define the link function of any of the parameters the `checklink()` function is used. This function takes four arguments.

`which.link:` which parameter the link is for, e.g. `"mu.link"`

`which.dist:` the current distribution, e.g. `"Normal"` (the name is only used to report an error in the specification of the link function)

`link:` which link is currently used, (the default value is the one given as arguments in the function definition, e.g. `substitute(mu.link)`will do the job)

`link.List:` the list of the possible links for the specific parameter, e.g. `c("inverse", "log", "identity")`

The available links to choose from are currently the ones used by the `make.link.gamlss()` function. This list includes:
`"logit"`, `"probit"`, `"cloglog"`, `"cauchit"` `"identity"`, `"log"`, `"sqrt"`, `"1/mu^2"`, `"mu^2"`, `"inverse"`, `"logshiftto1"`, `"logshiftto2"`, `"logshiftto0"`, `"inverse"` and `"own"`.
This may change in future **gamlss** releases to incorporate more link functions. For the use of the `own` see the help files under the `make.link.gamlss` where an example is given. [The object returned by `checklink()` contains the link function as a function of the current parameter, the inverse link function as a function of the current linear predictor and finally the first derivative of the inverse link function as a function of the linear predictor, i.e. d`mu`/d`eta`. These functions are used in the fitting of the distribution.]

The fitting information

The fitting algorithm uses the following information.

```
structure(list(family = c("NO", "Normal"),
           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(y, mu, sigma) (1/sigma^2) * (y - mu),
               d2ldm2 = function(sigma) -(1/sigma^2),
                 dldd = function(y, mu,sigma) ((y - mu)^2 - sigma^2)/(sigma^3),
               d2ldd2 = function(sigma) -(2/(sigma^2)),
              d2ldmdd = function(y) rep(0, length(y)),
           G.dev.incr = function(y, mu, sigma, ...) -2 * dNO(y, mu, sigma,
                                log = TRUE),
                rqres = expression( rqres(pfun = "pNO", 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) TRUE,
         sigma.valid = function(sigma) all(sigma > 0),
             y.valid = function(y) TRUE),
```

Here is an explanation of what all those entries mean:

- `family`: the name of the distribution, usually an abbreviated version and a more explicit one
- `parameters`: a list indicating whether the parameter will be fitted i.e. `mu=TRUE`, or fixed at initial values, e.g. `nu=FALSE`.
- `nopar`: the number of parameters
- `type`: the type of distribution i.e. "Continuous", "Discrete" or "Mixed"
- `mu.link, sigma.link`: the current link functions as character strings
- `mu.linkfun, sigma.linkfun`: the actual link functions returned from `checklink()`
- `mu.linkinv, sigma.linkinv`: the actual inverse link functions returned from `checklink()`
- `mu.dr, sigma.dr`: the actual first derivative of the inverse link functions returned from `checklink()`
- `dldm`: the first derivative of the likelihood with respect to the location parameter `mu`
- `d2ldm2`: the expected second derivative of the likelihood with respect to the location parameter `mu`
- `dldd`: the first derivative of the likelihood with respect to the scale parameter `sigma`
- `d2ldd2`: the expected second derivative of the likelihood with respect to the scale parameter `sigma`
- `d2ldmddd`: the expected cross derivative of the likelihood with respect to both the location `mu` and scale parameter `sigma`
- `G.dev.incr`: the global deviance (equal to minus twice the log likelihood)
- `rqres`: the definition of the (normalised quantile) residuals [Note these are randomized for discrete distributions], this requires specification of the type of the distribution
- `mu.initial, sigma.initial`: the default initial starting values for `mu` and `sigma` (both vectors of length n) for starting the algorithm
- `mu.valid, sigma.valid, y.valid`: valid range of values for the parameters (`mu` and `sigma`) and the response variable

Note that all the items above are compulsory. The expected second derivatives can be replaced in some cases by the negative squared first derivatives. [This can be done by using the expression `eval.parent(expression(-dldp^2))`]. Similarly the expected cross derivatives can be replaced in some cases by the negative cross product of the first derivatives.

The S3 class definition

Each family is defined as a `gamlss.family` object.

```
          class = c("gamlss.family", "family"))
```

The definition of the `d`, `p`, `q`, and `r` functions

```
dNO<-function(y, mu=0, sigma=1, log=FALSE)
 {
    fy <- dnorm(y, mean=mu, sd=sigma, log=log)
    fy
  }
pNO <- function(q, mu=0, sigma=1, lower.tail = TRUE, log.p = FALSE)
  {
          if (any(sigma <= 0))  stop(paste("sigma must be positive", "\n", ""))
    cdf <- pnorm(q, mean=mu, sd=sigma, lower.tail = lower.tail, log.p = log.p)
    cdf
   }

qNO <- function(p, mu=0, sigma=1, lower.tail = TRUE, log.p = FALSE)
  { if (any(sigma <= 0))  stop(paste("sigma must be positive", "\n", ""))
    if (log.p==TRUE) p <- exp(p) else p <- p
    if (any(p < 0)|any(p > 1))  stop(paste("p must be between 0 and 1", "\n",
                                           ""))
    q <- qnorm(p, mean=mu, sd=sigma, lower.tail = lower.tail )
    q
   }

rNO <- function(n, mu=0, sigma=1)
  {
    if (any(sigma <= 0))  stop(paste("sigma must be positive", "\n", ""))
    r <- rnorm(n, mean=mu, sd=sigma)
    r
  }
#------------------------------------------
```

These four functions [`dNO`, `pNO`, `qNO` and `rNO`] defined in general, the pdf, cdf, inverse cdf and random generating functions for the distribution. In the specific case of the normal distribution these function are not necessarily needed since `R` provides the equivalent functions `dnorm`, `pnorm`, `qnorm` and `rnorm`. We have included them here for convenience and consistency (with our parametrization of the distribution according to `mu` and `sigma`). From these four functions only the `d` function is usually used within the fitting function of a distribution while the `p` function is needed for calculating (and plotting) the residuals. The `d` function is used in the definition of global deviance and the `p` function in the definition of the normalized quantile residuals. The residuals are defined with the element `rqres` of the structure above which uses the function `rqres()` of the package (gamlss). The function `rqres()` needs to know what type of `gamlss.family` distribution we are using. For example for the `NO` distribution above we use the code `rqres(pfun="pNO", type="Continuous", y=y, mu=mu, sigma=sigma)`. This in effect will define the residuals as `qnorm(pNO(y,mu,sigma))`. For discrete distributions the function `rqres()` will randomized the residuals. For example the code for the Poisson distribution is `rqres(pfun="pPO", type="Discrete", ymin=0, y=y, mu=mu)`.

An Example: re-parametrising the NO distribution

As an example in which a different parametrization a distribution is required consider the parametrized normal distribution in which `mu` is still the mean but `sigma` is now the variance of the distribution rather the standard error. Only the changes from the previous definition of the function are printed here.

```
              NO2 <- function (mu.link ="identity", sigma.link="log")
...

          list(family = c("NO2","Normal with variance"),
...
                 dldm = function(y,mu,sigma) (1/sigma)*(y-mu),
               d2ldm2 = function(sigma) -(1/sigma),
                 dldd = function(y,mu,sigma)  0.5*((y-mu)^2-sigma)/(sigma^2),
               d2ldd2 = function(sigma)  -(1/(2*sigma^2)),
              d2ldmdd = function(y) rep(0,length(y)),
          G.dev.incr  = function(y,mu,sigma,...) -2*dNO2(y,mu,sigma,log=TRUE),
                rqres = expression(rqres(pfun="pNO2", type="Continuous",
                           y=y, mu=mu, sigma=sigma)),
...
}
```

The `d`, `p`, `q` and `r` functions have to be amended accordingly. Since `R` provides `d`, `p`, `q` and `r` functions for the normal distributions [given by `dnorm`, `pnorm`, `qnorm` and `rnorm` respectively] the amendment here can be easily done as follows:

```
dNO2<-function(y, mu=0, sigma=1, log=FALSE)
 {
    if (any(sigma <= 0))  stop(paste("sigma must be positive", "\n", ""))
    fy <- dnorm(y, mean=mu, sd=sqrt(sigma), log=log)
    fy
  }
pNO2 <- function(q, mu=0, sigma=1, lower.tail = TRUE, log.p = FALSE)
  {
    if (any(sigma <= 0))  stop(paste("sigma must be positive", "\n", ""))
    cdf <- pnorm(q, mean=mu, sd=sqrt(sigma), lower.tail = lower.tail,
                  log.p = log.p)
    cdf
   }
qNO2 <- function(p, mu=0, sigma=1, lower.tail = TRUE, log.p = FALSE)
  { if (any(sigma <= 0))  stop(paste("sigma must be positive", "\n", ""))
    if (log.p==TRUE) p <- exp(p) else p <- p
    if (any(p < 0)|any(p > 1))
       stop(paste("p must be between 0 and 1", "\n", ""))
    q <- qnorm(p, mean=mu, sd=sqrt(sigma), lower.tail = lower.tail )
    q
   }
rNO2 <- function(n, mu=0, sigma=1)
```

```
  {
    if (any(sigma <= 0))  stop(paste("sigma must be positive", "\n", ""))
    r <- rnorm(n, mean=mu, sd=sqrt(sigma))
    r
  }
```

More generally if an equivalent function does not exist it has to be written explicitly. For example this is another version of `dNO2`:

```
dNO2<-function(y, mu=0, sigma=1, log=FALSE)
 {
    if (any(sigma <= 0))  stop(paste("sigma must be positive", "\n", ""))
    loglik <- -0.5*log(2*pi*sigma)-0.5*((y-mu)^2)/sigma
    fy <- if(log==FALSE)  exp(loglik) else  loglik
    fy
  }
```

For users who would like to implement a different (or their own) distribution from the ones in Tables 6.1 and 6.2 the advice is to take one of the current distribution definition files (with the same number of parameters) and amend it. The `GU()`, `TF()`, and `BCT()` distributions are good examples of 2, 3, and 4 parameter continuous distributions respectively. `IG()` provides a good example where the `p` and `q` functions are calculated using numerical methods. The `PO()`, `NBI()` and `SI()` are good examples of 1, 2 and 3 parameter discrete distributions respectively. The `BB()` provides a example where the `p` and `q` functions are calculated using numerical methods. The `SICHEL()` distribution provides an example where numerical derivative are used implemented using the function `numeric.deriv()`.

6.5 The link functions

There are two functions in **gamlss** packages which are related to link functions of the parameters, the `make.link.gamlss()` and the `show.link()`. The first creates the link functions while the second displays them. Table **??** shows the usual link functions within the `gamlss` packages according to the range of the distribution parameter. The user can also create their own link function as we will show below.

range of parameters	link functions
$-\infty$ to $+\infty$	`identity`
0 to $+\infty$	`log`, `inverse`, `sqrt`, '`1/mu^2`', '`mu^2`'
0 to 1	`logit`, `probit`, `cauchit`, `cloglog`
1 to $+\infty$	`logshiftto1`
2 and $+\infty$	`logshiftto2`
0.00001 and $+\infty$	`logshiftto0` or `Slog` [1]

Table 6.4: The usual link functions available within the `gamlss` packages according to the range of the distribution parameters

The default link functions can be find by type the name of distribution. For example:

```
GA()
```

```
##
## GAMLSS Family: GA Gamma
## Link function for mu    : log
## Link function for sigma: log
```

is indicates that the default links for μ and σ are "log" links. Each link function requires the definition of four separate functions:

`linkfun(mu)`: the link function defining the predictor η, (`eta`), as a function of the current distribution parameter (which is always referred as `mu`) i.e. $\eta = g(\mu)$.

`linkinv(eta)`: the inverse of the link function as a function of the predictor η, (`eta`), i.e. $\mu = g^{-1}(\eta)$

`mu.eta(eta)`: the first derivative of the inverse link with respect to η, (`eta`), i.e. $\frac{d\mu}{d\eta}$

`validate(eta)`: in which range the values of η (`eta`) are defined

own link function

There are two ways for the user to create their own link functions within **gamlss**. The first one is by creating a new function having the right link information. This is the newest and recommended way. The other is by using the `own` link facility. That was the original way to generate a link function, but it is not as flexible as it can only be used to change the link of one parameters of the current distribution.

To demonstrate the use of link function we are using a binomial response variable example using the `aep` data. First we use `own` facility to create a complementary log-log link function $\eta = \log[-\log(1-\mu)]$ and we compare the results by using the existing `cloglog`.

```
# Try the complementary log-log function
# by using the Gumbel inverse cumulative distribution function
own.linkfun <- function(mu) { qGU(p=mu)}
own.linkinv <- function(eta) {
              thresh <- -qGU(.Machine$double.eps)
               eta <- pmin(thresh, pmax(eta, -thresh))
              pGU(eta)}
own.mu.eta <- function(eta) pmax(dGU(eta), .Machine$double.eps)
own.validate <- function(eta) TRUE
# h1 should be identical to cloglog in h2
 h1<-gamlss(y~ward+loglos+year, family=BI(mu.link="own"), data=aep)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 9456.145
## GAMLSS-RS iteration 2: Global Deviance = 9456.145
```

```
 h2<-gamlss(y~ward+loglos+year, family=BI(mu.link="cloglog"), data=aep)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 9456.145
## GAMLSS-RS iteration 2: Global Deviance = 9456.145
```

Note that the Gumbel distribution is a negatively skew distribution while the Reverse Gumbel (a reflation of the Gumbel) is positively skew. As a result a link function created using the Gumbel distribution will cause the binomial probability μ in `BI(N,`μ`)` to increa rapidly with η when $\mu > 0.5$ than when $\mu < 0.5$, while the one using the Reverse Gumbel will reverse this.

new link function

Here we create a link function based on the Reverse Gumbel i.e. $\eta = \log(-\log\mu)$ and compare the results with the `cloglog`.

```
# creating a  log-log link
loglog  <- function()
{
linkfun <- function(mu) { qRG(p=mu)}
linkinv <- function(eta) {
              thresh <- -qRG(.Machine$double.eps)
               eta <- pmin(thresh, pmax(eta, -thresh))
              pRG(eta)}
mu.eta <- function(eta) pmax(dRG(eta), .Machine$double.eps)
valideta <- function(eta) TRUE
link <- "loglog"
structure(list(linkfun = linkfun, linkinv = linkinv, mu.eta = mu.eta,
        valideta = valideta, name = link), class = "link-gamlss")
}
# fitting a model
h3<-gamlss(y~ward+loglos+year, family=BI(mu.link=loglog()), data=aep)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 9439.48
## GAMLSS-RS iteration 2: Global Deviance = 9439.48
```

```
AIC(h1,h2,h3, k=0)
```

```
##    df      AIC
## h3  5 9439.480
## h1  5 9456.144
## h2  5 9456.144
```

It is obvious that the log-log link function improves the global deviance.

Chapter 7

Finite mixture distributions

This chapter covers finite mixtures within GAMLSS in particular:

1. Finite mixtures with no parameters in common
2. Finite mixtures with several parameters in parameters in common

This chapter is important for fitting multimodal distributions to data.

7.1 Introduction to finite mixtures

This Chapter needs a major revision and connection to random effect Chapter.

Suppose that the random variable Y comes from component k, having probability (density) function $f_k(y)$, with probability π_k for $k = 1, 2, \ldots, K$, then the (marginal) density of Y is given by

$$f_Y(y) = \sum_{k=1}^{K} \pi_k f_k(y) \tag{7.1}$$

where $0 \leq \pi_k \leq 1$ is the prior (or mixing) probability of component k, for $k = 1, 2, \ldots, K$ and $\sum_{k=1}^{K} \pi_k = 1$. Note that the cumulative distribution function of Y will have a similar form and will be:

$$F_Y(y) = \sum_{k=1}^{K} \pi_k F_k(y). \tag{7.2}$$

More generally the probability (density) function $f_k(y)$ for component k may depend on parameters $\boldsymbol{\theta}_k$ and explanatory variables $\mathbf{x}_k$, i.e. $f_k(y) = f_k(y|\boldsymbol{\theta}_k, \mathbf{x}_k)$. Hence $f_Y(y)$ depends on parameters $\boldsymbol{\psi} = (\boldsymbol{\theta}, \boldsymbol{\pi})$ where $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_K)$ and $\boldsymbol{\pi}^T = (\boldsymbol{\pi}_1, \boldsymbol{\pi}_2, \ldots, \boldsymbol{\pi}_K)$ and explanatory variables $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_K)$, i.e. $f_Y(y) = f_Y(y|\boldsymbol{\psi}, \mathbf{x})$, and

$$f_Y(y|\boldsymbol{\psi}, \mathbf{x}) = \sum_{k=1}^{K} \pi_k f_k(y|\boldsymbol{\theta}_k, \mathbf{x}_k) \tag{7.3}$$

Subsequently we omit the conditioning on $\boldsymbol{\theta}_k$, $\mathbf{x}_k$ and $\boldsymbol{\psi}$ to simplify the presentation.

In general finite mixture distributions are fitted within GAMLSS using the EM algorithm. Certain specific mixtures distributions are explicitly available in **gamlss** packages. In particular the zero adjusted gamma (`ZAGA`), the zero adjusted inverse Gaussian (`ZAIG`), and the four parameter beta inflated at zero and one (`BEINF`), and a variety of zero inflated and adjusted discrete distributions (`ZIP`, `ZIP2`, `ZAP`, `ZINBI`, `ZANBI`, `ZIPIG`, `ZIBI`, `ZIBB`).]

In Sections 7.2, 7.3 and 7.4 we consider respectively maximum likelihood estimation, the corresponding fitting function `gamlssMX` and examples for finite mixtures models with **no** parameters in common, while in Sections 7.5, 7.6 and 7.7 we consider respectively maximum likelihood estimation, the corresponding fitting function `gamlssNP` and examples for finite mixture models with parameters in common. Throughout this chapter we will assume that all K components of the mixture can be represented by GAMLSS models.

7.2 Finite mixtures with no parameters in common

Here the parameter sets $(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_k)$ are distinct, i.e. no parameter is common to two or more parameters sets. Note that what this means in practice within GAMLSS is that the conditional distribution components in (7.1), $f_k(y)$, can have different `gamlss.family` distributions, e.g. one can be `GA` and the other `IG`.

7.2.1 The likelihood function

Given n independent observations y_i for $i = 1, 2, \ldots, n$, from finite mixture model (7.3), the likelihood function is given by

$$L = L(\boldsymbol{\psi}, \mathbf{y}) = \prod_{i=1}^{n} f_{Y_i}(y_i) = \prod_{i=1}^{n} \left[\sum_{k=1}^{K} \pi_k f_k(y_i) \right] \tag{7.4}$$

where $\mathbf{y} = (y_1, y_2, \ldots, y_n)$, $f_k(y_i) = f_k(y_i|\boldsymbol{\theta}_k, \mathbf{x}_{ki})$, with log likelihood function given by

$$\ell = \ell(\boldsymbol{\psi}, \mathbf{y}) = \sum_{i=1}^{n} \log \left[\sum_{k=1}^{K} \pi_k f_k(y_i) \right] \tag{7.5}$$

We wish to maximize ℓ with respect to $\boldsymbol{\psi}$, i.e. with respect to $\boldsymbol{\theta}$ and $\boldsymbol{\pi}$. The problem is that the log function between the two summations in (7.5) makes it difficult. One solution, especially for simple mixtures where no explanatory variables are involved, is to use a numerical maximization technique, e.g. function `optim` in R, to maximize the log likelihood in (7.5) numerically, see for example Venables and Ripley [2002] Chapter 16. More generally an EM algorithm can be used to maximize (7.5).

7.2.2 Maximizing the likelihood function using the EM algorithm

Here we will use the EM algorithm, (Dempster, A., Laird, N. and Rubin [1977]) to maximize (7.5) with respect to $\boldsymbol{\psi}$, treating all the component indicator variables (i.e $\boldsymbol{\delta}$, defined below) as missing variables.

Let

$$\delta_{ik} = \begin{cases} 1, & \text{if observation } i \text{ comes from component } k \\ 0, & \text{otherwise} \end{cases} \tag{7.6}$$

for $k = 1, 2, \ldots, K$ and $i = 1, 2, \ldots, n$. Let $\boldsymbol{\delta}_i^T = (\delta_{i1}, \delta_{i2}, \ldots, \delta_{ik})$ be the indicator vector for observation i. If observation i comes from component k then $\boldsymbol{\delta}_i$ is a vector of zeros, except for the k^{th} value which is $\delta_{ik} = 1$. Let $\boldsymbol{\delta}^T = (\boldsymbol{\delta}_1^T, \boldsymbol{\delta}_2^T, \ldots, \boldsymbol{\delta}_n^T)$ combine all the indicator variable vectors. Then the complete data, i.e. observed $\mathbf{y}$ and unobserved $\boldsymbol{\delta}$, has complete likelihood function given by

$$\begin{aligned} L_c = L_c(\boldsymbol{\psi}, \mathbf{y}, \boldsymbol{\delta}) &= f(\mathbf{y}, \boldsymbol{\delta}) = \prod_{i=1}^{n} f(y_i, \boldsymbol{\delta}_i) \\ &= \prod_{i=1}^{n} f(y_i|\boldsymbol{\delta}_i) f(\boldsymbol{\delta}_i) \\ &= \prod_{i=1}^{n} \left\{ \prod_{k=1}^{K} \left[f_k(y_i)^{\delta_{ik}} \pi_k^{\delta_{ik}} \right] \right\}, \end{aligned} \tag{7.7}$$

since if $\delta_{ik} = 1$ and $\delta_{ik'} = 0$ for $k^{'} \neq k$, then

$$\begin{aligned} f(y_i|\boldsymbol{\delta}_i) f(\boldsymbol{\delta}_i) &= f_k(y_i)\pi_k, \\ &= f_k(y_i)^{\delta_{ik}} \pi_k^{\delta_{ik}} \\ &= \textstyle\prod_{k=1}^{K} f_k(y_i)^{\delta_{ik}} \pi_k^{\delta_{ik}} \end{aligned}$$

and hence $f(y_i|\boldsymbol{\delta}_i) f(\boldsymbol{\delta}_i) = \prod_{k=1}^{K} f_k(y_i)^{\delta_{ik}} \pi_k^{\delta_{ik}}$ for all $\boldsymbol{\delta}_i$.

From (7.7) the complete log likelihood is given by

$$\ell_c = \ell_c(\boldsymbol{\psi}, \mathbf{y}, \boldsymbol{\delta}) = \sum_{i=1}^{n} \sum_{k=1}^{K} \delta_{ik} \log f_k(y_i) + \sum_{i=1}^{n} \sum_{k=1}^{K} \delta_{ik} \log \pi_k \tag{7.8}$$

If $\boldsymbol{\delta}$ were known then, since $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots \boldsymbol{\theta}_K$ have no parameter in common, ℓ_c could be maximized over each $\boldsymbol{\theta}_k$ separately, since the likelihood separates.

The EM algorithm alternates between the E-step and the M-step until convergence. Iteration $(r+1)$ of the EM algorithm comprises an E-step followed by an M-step.

E-step

At the $(r+1)^{th}$ iteration, the E-step finds Q, the conditional expectation of the complete data log likelihood (7.8), over the missing $\boldsymbol{\delta}$, given $\mathbf{y}$ and the current parameter estimates $\hat{\boldsymbol{\psi}}^{(r)}$ from iteration r.

M-step

At the $(r+1)^{th}$ iteration, the M step maximizes Q with respect to $\boldsymbol{\psi}$, where Q is the conditional expected value from E-step.

How does it work in practice we have to explain.

7.2.3 Modelling the mixing probabilities

Here we extend the finite mixture model by assuming that the mixing probabilities $\boldsymbol{\pi}_k$ for $k = 1, 2, \ldots, K$ for observations $i = 1, 2, \ldots, n$ are not fixed constants but depend on explanatory variables $\mathbf{x}_0$ and parameters $\boldsymbol{\alpha}$, and hence depend on i, so $f_{Y_i}(y_i) = \sum_{k=1}^{K} \pi_{ik} f_k(y_i)$. We model the mixing probabilities π_{ik} using a multinomial logistic model where $\boldsymbol{\delta}_i$ is a single draw from a multinomial distribution with probability vector $\boldsymbol{\pi}$, i.e. $\boldsymbol{\delta}_i \sim M(1, \boldsymbol{\pi})$ and

$$\log\left[\frac{\pi_{ik}}{\pi_{iK}}\right] = \boldsymbol{\alpha}_k^T \mathbf{x}_{0i} \tag{7.9}$$

for $k = 1, 2, \ldots, K$ and $i = 1, 2, \ldots, n$. Hence

$$\pi_{ik} = \frac{\exp\left\{\boldsymbol{\alpha}_k^T \mathbf{x}_{0i}\right\}}{\sum_{k=1}^{K} \exp\left\{\boldsymbol{\alpha}_k^T \mathbf{x}_{0i}\right\}} \tag{7.10}$$

for $k = 1, 2, \ldots, K$ and $i = 1, 2, \ldots, n$ where $\boldsymbol{\alpha_K} = \mathbf{0}$. Consequently the complete log likelihood is given by replacing π_k by π_{ik} in equation (7.8) to give

$$\ell_c = \ell_c(\boldsymbol{\psi}, \mathbf{y}, \boldsymbol{\delta}) = \sum_{i=1}^{n}\sum_{k=1}^{K} \delta_{ik} \log f_k(y_i) + \sum_{i=1}^{n}\sum_{k=1}^{K} \delta_{ik} \log \pi_{ik} \tag{7.11}$$

7.2.4 Zero components

Special cases of the models described above are distributions which we described earlier as type mixed. For example, the zero adjusted inverse Gaussian distribution (ZAIG) can be thought of as a finite mixture where the first component is identically zero, i.e. $y = 0$, with probability 1. Hence

$$f_1(y) \quad = \quad \begin{cases} 1, & \text{if y=0} \\ 0, & \text{otherwise.} \end{cases} \tag{7.12}$$

The second component is an inverse Gaussian distribution. Distributions of this type can be also fitted with the EM algorithm described in the previous section.

7.3 The gamlssMX() function

The function to fit finite mixtures with no parameters in common is `gamlssMX()`. In this section we describe how it works. Examples of using the function are given in the next section. The function `gamlssMX()` has the following arguments:

formula This argument should be a single formula (or a list of formulae of length K the number of components in the mixture) for modelling the predictor for the μ parameter of the model. If a single formula is used then the K mixture components have the same predictor for μ, but different parameters in their predictors (since there are no parameters in common to two or more of the K components). Note that modelling the rest of the distributional parameters can be done by using the usual `gamlss()` formula arguments, e.g. `sigma.fo=~x` , which passes the arguments to `gamlss()`. Again either a single common formula or a list of formula of length K is used.

pi.formula This should be a formula for modelling the predictor for prior (or mixing) probabilities as a function of explanatory variables in the multinomial model (7.9). The default model is constants for the prior (or mixing) probabilities. Note that no smoothing or other additive terms are allowed here, only the usual linear terms. The modelling here is done using the `multinom()` function from package `nnet`.

family This should be a `gamlss.family` distribution (or a list of K distributions). Note that if different distributions are used here, it is preferable (but not essential) that their parameters are comparable for ease of interpretation.

weights For declaring prior weights if needed.

K For declaring the number of components in the finite mixture with default `K=2`

prob For setting starting values for the prior probabilities.

data The data frame containing the variables in the fit. Note that this is compulsory if `pi.formula` is used for modelling the prior (or mixing) probabilities.

control This argument sets the control parameters for the EM iterations algorithm. The default setting are given in the `MX.control` function

g.control This argument can be used to pass to `gamlss()` control parameters, as in `gamlss.control`.

zero.component This argument declares whether or not there is a zero component, i.e. y identically equal to zero, $y = 0$, in the finite mixture.

... For extra arguments to be passed to `gamlss()`.

What the output produce Fitted values? residuals? we have to explain

7.4 Examples using the gamlssMX() function

7.4.1 The Old Faithful geyser data

The data on the Old Faithful geyser has two variables, `duration`, the duration of the eruption and `waiting`, the waiting time in minutes until the next eruption. Firstly, the variable `waiting` is used on its own to demonstrate the fitting of a finite mixture to a single response variable. In the second part the data are modified and used as to model of the mixture response variable against an explanatory variable.

Fitting a finite mixture to a single response

Data summary: the old faithful geyser

R data file: `geyser` in package **MASS** of dimensions 299×2

variables

`waiting` : the waiting time (in minutes) until the next eruption.

`duration` : the duration of the eruption.

purpose: only the variable `waiting` is used here to demonstrate the fitting of a finite mixture.

conclusion: A two component inverse Gaussian distribution is found to be suitable

Here we study the `waiting` time on its own. We use `waiting` time to demonstrate how to fit a variety of two component mixtures of continuous distributions and then select the 'best' using AIC. Two component mixtures of normal, gamma, reverse Gumble, Gumble, logistic and inverse Gaussian distributions are fitted:

```
data(geyser)
set.seed(1581)
mNO <- gamlssMX(waiting ~ 1, data = geyser, family = NO, K = 2)
mGA <- gamlssMX(waiting ~ 1, data = geyser, family = GA, K = 2)
mRG <- gamlssMX(waiting ~ 1, data = geyser, family = RG, K = 2)
mGU <- gamlssMX(waiting ~ 1, data = geyser, family = GU, K = 2)
mLO <- gamlssMX(waiting ~ 1, data = geyser, family = LO, K = 2)
mIG <- gamlssMX(waiting ~ 1, data = geyser, family = IG, K = 2)
AIC(mNO, mGA, mRG, mGU, mLO, mIG)
```

```
##     df      AIC
## mIG  5 2321.827
## mGA  5 2322.764
## mRG  5 2323.879
## mNO  5 2325.084
## mLO  5 2328.147
## mGU  5 2420.051
```

```
mIG
```

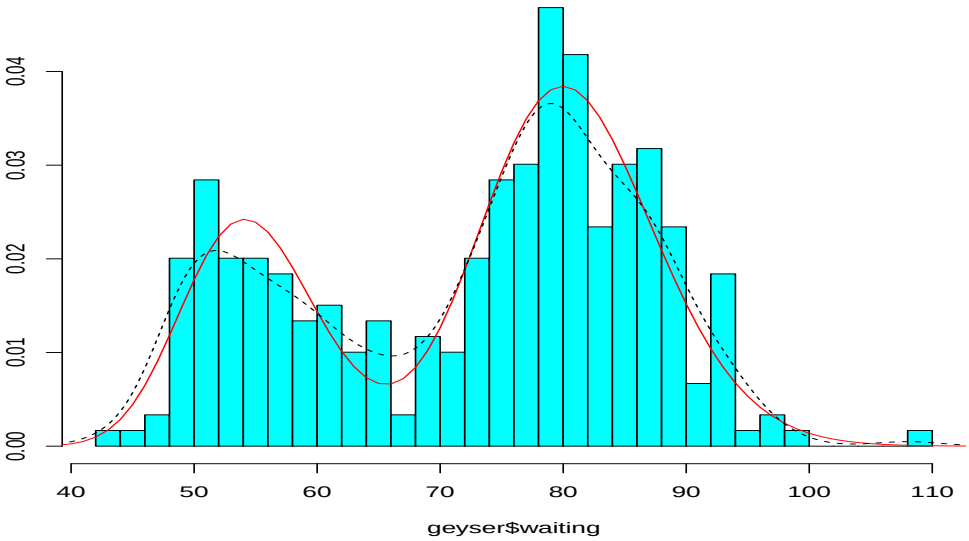
```
##
## Mixing Family:  c("IG", "IG")
##
## Fitting method: EM algorithm
##
## Call:  gamlssMX(formula = waiting ~ 1, family = IG, K = 2, data = geyser)
##
##
## Mu Coefficients for model: 1
## (Intercept)
##       4.393
## Sigma Coefficients for model: 1
## (Intercept)
##      -4.642
## Mu Coefficients for model: 2
## (Intercept)
##       4.006
## Sigma Coefficients for model: 2
## (Intercept)
##      -4.304
##
```

```
## Estimated probabilities: 0.669591 0.330409
##
## Degrees of Freedom for the fit: 5 Residual Deg. of Freedom   294
## Global Deviance:     2311.83
##              AIC:     2321.83
##              SBC:     2340.33
```

The best model appears to be `mIG`, the two component inverse Gaussian (IG) model for $Y(=waiting)$ given by $f_Y(y) = \hat{\pi}_i f_1(y) + \hat{\pi}_i f_2(y) = 0.67 f_1(y) + 0.33 f_2(y)$ where $f_1(y)$ is an inverse Gaussian distribution, $IG(\mu_1, \sigma_1)$ with $\hat{\mu}_1 = \exp(4.393) = 80.88$ and $\hat{\sigma}_1 = \exp(-4.641) = 0.009843$ and $f_2(y)$ is an inverse Gaussian distribution, $IG(\mu_2, \sigma_2)$ with $\hat{\mu}_2 = \exp(4.006) = 54.93$ and $\hat{\sigma}_2 = \exp(-4.304) = 0.01351$. We next plot a histogram of the data together with the fitted two component IG model (solid line) and a non-parametric density estimator (dash line).

```
truehist(geyser$waiting, h = 2)
fyIG <- dMX(y = seq(39, 115, 1), mu = list(exp(4.393), exp(4.006)),
     sigma = list(exp(-4.642), exp(-4.304)), pi = list(0.6695835,
         0.3304165), family = list("IG", "IG"))
lines(seq(39, 115, 1), fyIG, col = "red", lty = 1)
lines(density(geyser$waiting, width = "SJ-dpi"), lty = 2)
```

Figure 7.1



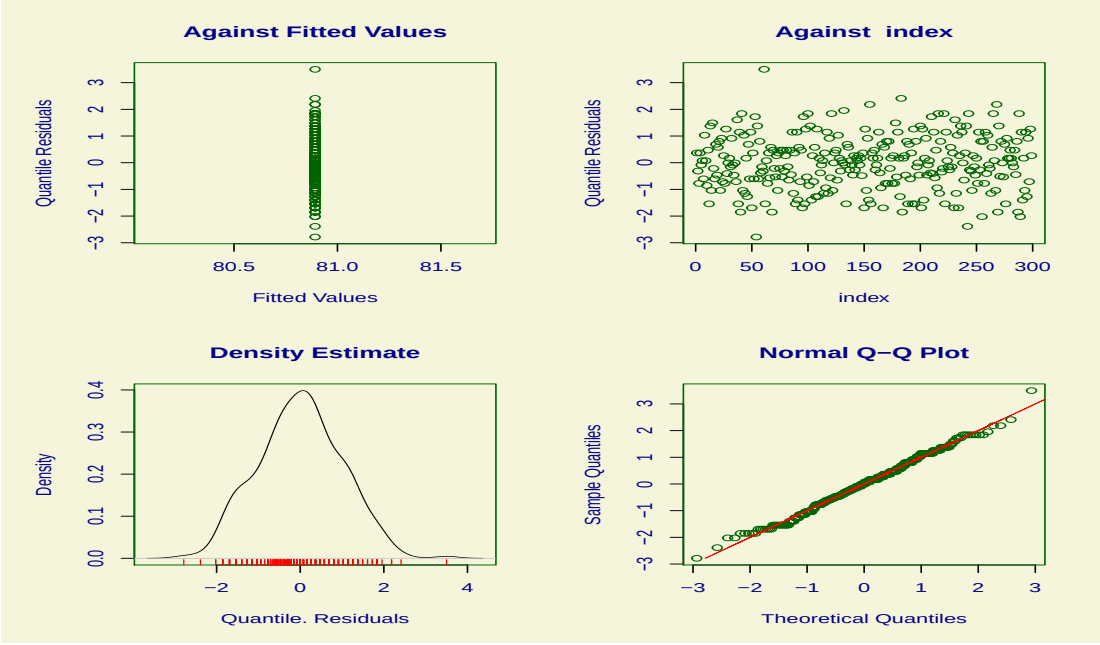
R code on page 159

Figure 7.1: A histogram of variable waiting time (to next eruption from the Old Faithful geyser data), together with a non-parametric density estimator (– – –) and the fitted two component IG model (___)

The residuals of the final fitted model `mIG` are plotted next.

Figure 7.2

```
plot(mIG)

## ****************************************************************
##   Summary of the Randomised Quantile Residuals
##                            mean   =  -0.001605
##                        variance   =  0.9946
##                coef. of skewness  =  0.07091
##                coef. of kurtosis  =  2.862
## Filliben correlation coefficient  =  0.9969
## ****************************************************************
```



R code on page 160

Figure 7.2: The residual plot from the fitted two component IG model for waiting time from the Old Faithful geyser data

Fitting a finite mixture to a simple regression problem

Data summary: the old faithful geyser

R data file: `geyser2` created from `gayser` of dimensions 298×2

variables

`waiting` : the response variable, waiting time (in minutes) until the next eruption.

`duration` : previous duration of the eruption (used as explanatory variable)

purpose: model the distribution of waiting time until the next eruption given the explana-

tory variable previous duration

conclusion: The response can be modeled as a mixture of two components each having an inverse Gaussian distribution or a single component inverse Gaussian with smoothing

We will now model the the `waiting` time until the next eruption as a function of the previous codeduration. We will also follow Venables and Ripley [2002] p441 and model the probabilities, π's, of belonging to one of the two mixture components as functions of the previous `duration` of the eruption.

We first create a data frame containing the `waiting` time to the next eruption and the previous `duration` of the eruption. The data are displayed in the left panel of Figure 7.3. Then we fit a normal (NO) two component mixture model, used by Venables and Ripley [2002], and a inverse Gaussian (IG) two component mixture model. First we fit constant models for the predictors of both μ and π, then include duration in the predictor of each of μ and π separately and finally include duration in the predictor of both μ and π. We compare all the models using AIC.

```
geyser2 <- matrix(0, ncol = 2, nrow = 298)
geyser2[, 1] <- geyser$waiting[-1]
geyser2[, 2] <- geyser$duration[-299]
colnames(geyser2) <- c("waiting", "duration")
geyser2 <- data.frame(geyser2)
set.seed(1581)
mNO1 <- gamlssMX(waiting ~ 1, data = geyser2, family = NO, K = 2)
mIG1 <- gamlssMX(waiting ~ 1, data = geyser2, family = IG, K = 2)
mNO2 <- gamlssMX(waiting ~ 1, pi.formula = ~duration, data = geyser2,
     family = NO, K = 2)
mIG2 <- gamlssMX(waiting ~ 1, pi.formula = ~duration, data = geyser2,
    family = IG, K = 2)
mNO3 <- gamlssMX(waiting ~ duration, pi.formula = ~1, data = geyser2,
    family = NO, K = 2)
mIG3 <- gamlssMX(waiting ~ duration, pi.formula = ~1, data = geyser2,
     family = IG, K = 2)
mNO4 <- gamlssMX(waiting ~ duration, pi.formula = ~duration,
     data = geyser2, family = NO, K = 2)
mIG4 <- gamlssMX(waiting ~ duration, pi.formula = ~duration,
     data = geyser2, family = IG, K = 2)
AIC(mNO1, mNO2, mNO3, mNO4, mIG1, mIG2, mIG3, mIG4)
```

```
##      df      AIC
## mIG4  8 1930.034
## mNO4  8 1936.679
## mNO3  7 1953.317
## mIG3  7 1961.234
## mIG2  6 1970.647
## mNO2  6 1981.932
## mIG1  5 2315.304
## mNO1  5 2318.472
```

```
mIG4
```

```
##
```

```
## Mixing Family:  c("IG", "IG")
##
## Fitting method: EM algorithm
##
## Call:  gamlssMX(formula = waiting ~ duration, pi.formula = ~duration,
##     family = IG, K = 2, data = geyser2)
##
## Mu Coefficients for model: 1
## (Intercept)     duration
##     4.09618      0.07007
## Sigma Coefficients for model: 1
## (Intercept)
##      -4.807
## Mu Coefficients for model: 2
## (Intercept)     duration
##      3.6312       0.1935
## Sigma Coefficients for model: 2
## (Intercept)
##      -4.351
## model for pi:
##          (Intercept)  duration
## fac.fit2    10.18838 -3.131291
##
## Estimated probabilities:
##          pi1        pi2
## 1 0.91598279 0.08401721
## 2 0.03058744 0.96941256
## 3 0.91187829 0.08812171
## ...
##
## Degrees of Freedom for the fit: 8 Residual Deg. of Freedom   290
## Global Deviance:     1914.03
##             AIC:     1930.03
##             SBC:     1959.61
```

Note that in order to model the π's, the function `gamlssMX` needs the `data` argument. The best model using AIC is model `mIG4`. This model is a mixture of two components. In each component waiting time has an inverse Gaussian distribution, with a simple linear regression model in duration for the predictor of the mean and a constant scale. The predictor for the mixing probability is also a simple linear regression model in duration. So the final mixture model `mIG4` is given by

$$f_Y(y) = \hat{\pi}_1 f_1(y) + \hat{\pi}_2 f_2(y)$$

where $f_1(y)$ is an inverse Gaussian distribution $IG(\hat{\mu}_1, \hat{\sigma}_1)$ with

$$\hat{\mu}_1 = \exp\{4.0962 + 0.07007 * \text{duration}\}$$

and

$$\hat{\sigma}_1 = \exp\{-4.807\} = 0.00817$$

and where $f_2(y)$ is also an inverse Gaussian distribution $IG(\hat{\mu}_2, \hat{\sigma}_2)$ with

$$\hat{\mu}_2 = \exp\{3.6312 + 0.1935 * \text{duration}\}$$

and

$$\hat{\sigma}_2 = \exp\{-4.351\} = 0.01289$$

and where

$$\log[\hat{\pi}_2/(1-\hat{\pi}_2)] = \boldsymbol{\eta}_\pi = 10.188 - 3.1313 * \text{duration}$$

.

Figure 7.3(a) plots the data together with the fitted means of each of the two components. Figure 7.3(b) shows the fitted probability of belonging to group 1. As the previous eruption duration increases, the probability that the waiting time will belong to component 1 increases. Figure 7.3 was obtained by the following commands:

Figure 7.3

```
op <- par(mfrow = c(1, 2))
plot(waiting ~ duration, data = geyser2, xlab = "previous duration",
     ylab = "waiting time", main = "(a)")
lines(fitted(mIG4$models[[1]])[order(geyser2$duration)] ~
geyser2$duration[order(geyser2$duration)],
     col = "dark green", lty = 3, lwd = 2)
lines(fitted(mIG4$models[[2]])[order(geyser2$duration)] ~
geyser2$duration[order(geyser2$duration)],
   col = "red", lty = 4, lwd = 2)
plot(mIG4$prob[, 1][order(duration)] ~ duration[order(duration)],
 data = geyser2, xlab = "previous duration", ylab = "probability of component 2",
     main = "(b)")
lines(mIG4$prob[, 1][order(duration)] ~ duration[order(duration)],
    data = geyser2)
lines(mIG4$prob[, 1][order(duration)] ~ duration[order(duration)],
     data = geyser2)
par(op)
```

Figure 7.4 shows the fitted distribution in three dimensions, where $f1$ is the fitted conditional probability density function for weighted time (using the commands below). Figure 7.6 (a) shows this as a `levelplot` (see later for the commands).

```
grid <- expand.grid(duration = seq(1.5, 5.5, 0.1), waiting = seq(40,
     110, 0.5))
etapi <- 10.19069 - 3.132215 * grid$duration
etamu1 <- 4.09618 + 0.07007 * grid$duration
etamu2 <- 3.6312 + 0.1935 * grid$duration
pp <- (exp(etapi)/(1 + exp(etapi)))
grid$f1 <- dMX(y = grid$waiting, mu = list(exp(etamu1), exp(etamu2)),
     sigma = list(exp(-4.807), exp(-4.351)), pi = list(1 - pp, pp),
     family = list("IG", "IG"))
```

Figure 7.4

```
library(lattice)
wireframe(f1 ~ duration * waiting, data = grid, aspect = c(1,
     0.5), drape = TRUE)
```

R code on page 163

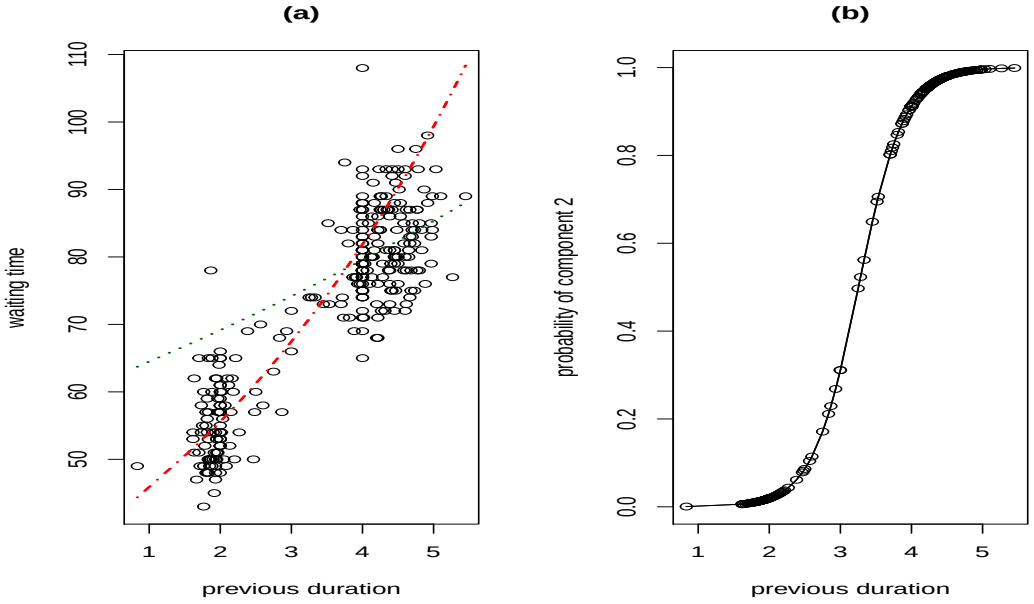
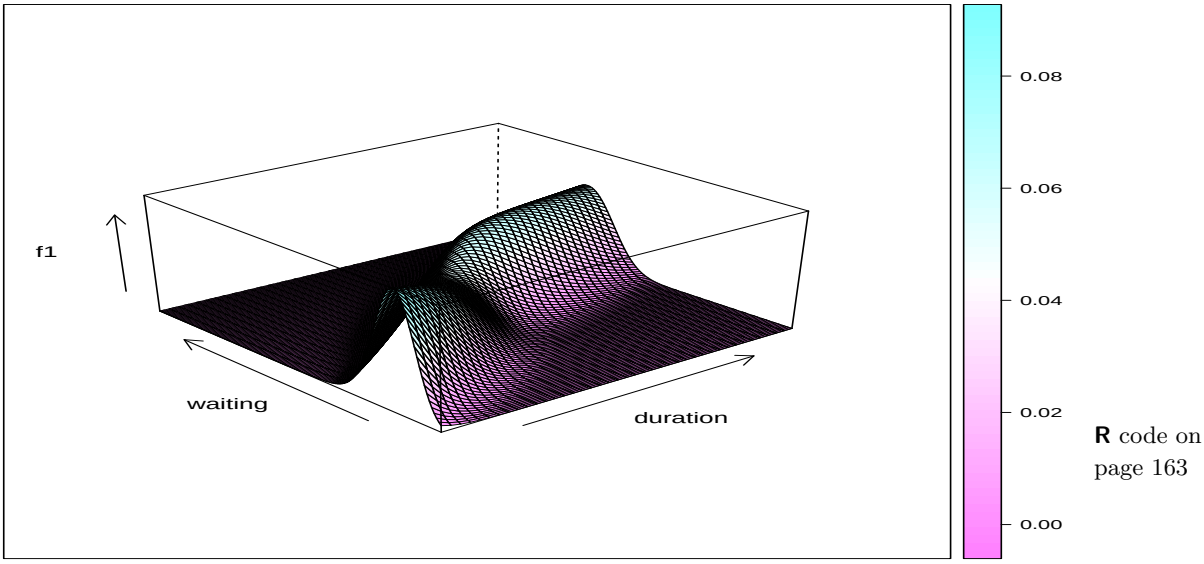


Figure 7.3: (a) A scatter plot of the waiting time (to next eruption)against the previous eruption duration from the Old Faithful geyser data together with the fitted values from the two components, (dotted and dashed for component 1 and 2 respectively) (b) a plot of the probability of belonging to component 1 as a function of duration, estimated from model `mIG4`



R code on page 163

Figure 7.4: Fitted conditional probability density function (`f1`) for waiting time to the next eruption given the previous eruption duration for model `mIG4`

Model `mIG4` provides us with an example of a regression model where the response variable has a mixture distribution with two components and where the probability of belonging to each component of the mixture is modelled as a function of a single explanatory variable. The model is appropriate if modelling the probability of belonging to a component is of interest. If, on the other hand, the interest lies in just modelling the waiting time as a function of the previous duration, a simple GAMLSS model could be appropriate.

We will try here to compare the `mIG4` (finite mixture) model with a single component model (not a mixture) using the inverse Gaussian distribution with regression models in the previous duration for both μ and σ. A flexible cubic smoothing spline function as a function of the previous duration is also used for μ and σ.

```
mIG5 <- gamlss(waiting ~ duration, sigma.formula = ~duration,
     data = geyser2, family = IG, trace = FALSE)
mIG6 <- gamlss(waiting ~ cs(duration), sigma.formula = ~duration,
     data = geyser2, family = IG, trace = FALSE)
mIG7 <- gamlss(waiting ~ cs(duration), sigma.formula = ~cs(duration),
     data = geyser2, family = IG, trace = FALSE)
mIG8 <- gamlss(waiting ~ cs(duration), sigma.formula = ~1, data = geyser2,
     family = IG, trace = FALSE)
AIC(mIG4, mIG5, mIG6, mIG7, mIG8)
```

```
##             df      AIC
## mIG6  7.000763 1928.512
```

```
## mIG4  8.000000 1930.034
## mIG7 10.000071 1933.061
## mIG8  6.000815 1957.221
## mIG5  4.000000 1958.542
```

Model `mIG6` is marginally better that model `mIG4` in terms of AIC. Figure 7.5 compares the fitted means for the two models. The smooth fitted mean line of model `mIG6` follows closely the component 2 line of model `mIG4` up to `duration` around 4 and then the component 1 line. The two models behave very similarly as far the mead model is concerned.

Figure 7.5

```
plot(waiting ~ duration, data = geyser2, xlab = "previous duration",
     ylab = "waiting time")
lines(fitted(mIG4$models[[1]])[order(geyser2$duration)] ~
 geyser2$duration[order(geyser2$duration)],
     col = "green", lty = 3, lwd = 1.5)
lines(fitted(mIG4$models[[2]])[order(geyser2$duration)] ~
 geyser2$duration[order(geyser2$duration)],
     col = "red", lty = 4, lwd = 1.5)
lines(fitted(mIG6)[order(duration)] ~ duration[order(duration)],
     data = geyser2, col = "blue", lty = 1, lwd = 1.5)
```

R code on page 166

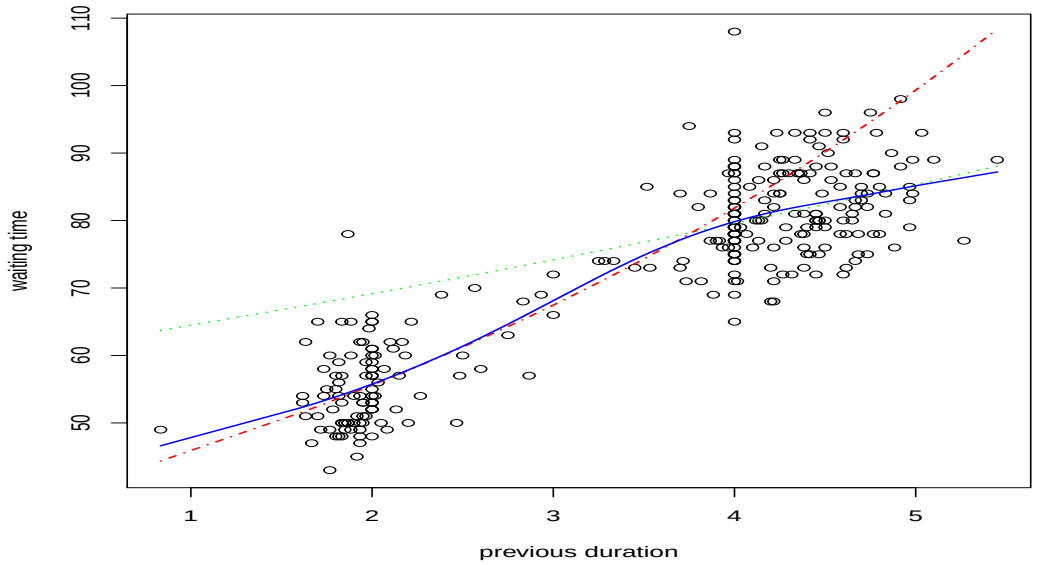


Figure 7.5: Comparison of the fitted values for μ for models `mIG4` (dashed and dotted lines) and `mIG6` (solid line)

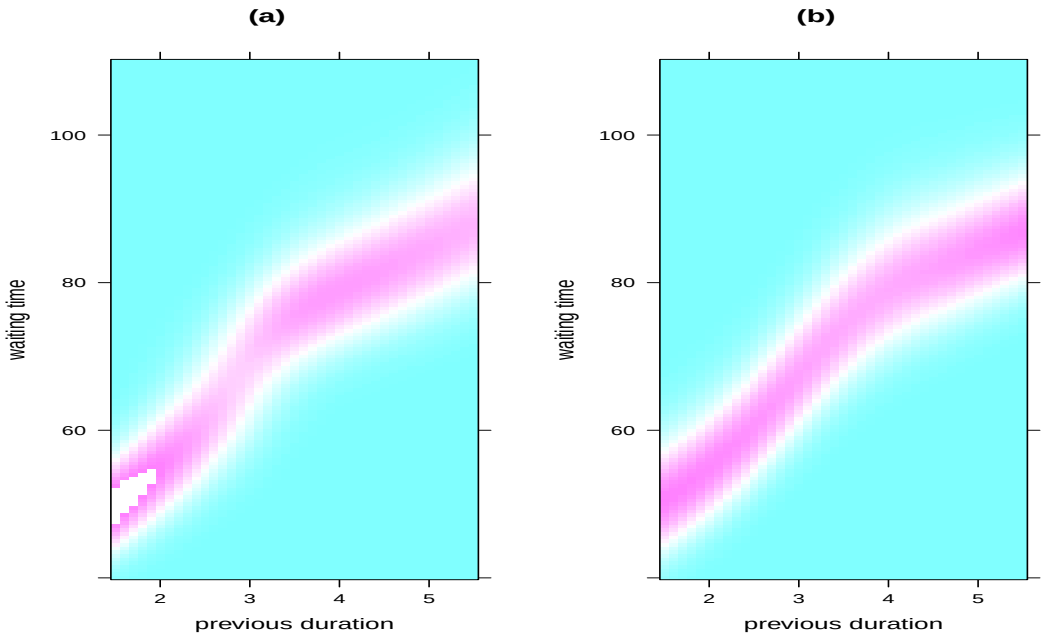
Figures 7.6 (a) and (b) show levelplots of the conditional probability density function (pdf) for waiting time given the previous eruption time for models (a) `mIG4` and (b) `mIG6` respectively obtained using the commands below. The plots are similar, although model `mIG4` has a higher

conditional pdf for waiting time to the next eruption around 50 minutes when previous duration is less than 2.

```
mu <- predict(mIG6, what = "mu", type = "response", newdata = grid[,
     c("waiting", "duration")], data = geyser2)

## new prediction

sigma <- predict(mIG6, what = "sigma", type = "response", newdata = grid[,
     c("waiting", "duration")], data = geyser2)
grid$f2 <- dIG(x = grid$waiting, mu = mu, sigma = sigma)
```

```
op <- par(mfrow = c(1, 2))
print(levelplot(f1 ~ duration * waiting, data = grid, colorkey = F,
     at = seq(0, 0.075, 0.001), xlab = "previous duration", ylab = "waiting time",
     col.regions = rev(trellis.par.get("regions")$col), main = "(a)"),
     split = c(1, 1, 2, 1), more = TRUE)
print(levelplot(f2 ~ duration * waiting, data = grid, colorkey = F,
     at = seq(0, 0.075, 0.001), xlab = "previous duration", ylab = "waiting time",
     col.regions = rev(trellis.par.get("regions")$col), main = "(b)"),
     split = c(2, 1, 2, 1))
par(op)
```

Figure 7.6



R code on page 167

Figure 7.6: Levelplot of the fitted conditional probability density function of the waiting time given the previous eruption time for models (a) `mIG4` and model (b) `mIG6`

7.5 Finite mixtures with parameters in common

Here the K components of the mixture may have parameters in common, i.e. the parameter sets $(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_k)$ are not disjoint. The prior (or mixing) probabilities can be either assumed to be constant or that may depend on explanatory variables $\mathbf{x}_0$ and parameters $\boldsymbol{\alpha}$ through a multinomial logistic model as in Section 7.2.3. Note however that in the implementation of the function `gamlssNP()`, which can be used fitting finite mixture with common components, the probabilities are assumed to be constant and are not depending on explanatory variables.

It is assumed throughout that the K components $f_k(y) = f_k(y|\boldsymbol{\theta}_k, \mathbf{x}_k)$ for $k = 1, 2, \ldots, K$ can be represented by GAMLSS models. However, since some of the parameters may be common in all K components, the distribution used must be the same for all K components. Similarly the link functions of the distribution parameters must be the same for all K components. In our notation in this Chapter, the parameter vector $\boldsymbol{\theta}_k$ contains all the parameters in the (linear) predictor models for μ, σ, ν and τ for component k, for $k = 1, 2, \ldots, K$. Here are some examples to clarify this.

Example 1, Mixture of K Poisson regression models: $f(y) = \sum_{k=1}^{K} \pi_k f_k(y)$ where $f_k(y)$ is $PO(\mu_k)$ for $k = 1, 2, \ldots, K$, and where $\log \mu_k = \beta_{ok} + \beta_1 x$. Here the slope parameter β_1, a predictor parameter for the distribution parameter μ_k, is the same for all K components, but the intercept parameter β_{ok} depends on k, for $k = 1, 2, \ldots, K$.

Example 2, Mixture of K negative binomials regression models: Let $f_k(y)$ be $NBI(\mu_k, \sigma_k)$ for $k = 1, 2, \ldots, K$, where $\log \mu_k = \beta_{10k} + \beta_{11} x$ and $\log \sigma_k = \log \sigma = \beta_{2o} + \beta_{21} x$. Here the predictor slope parameter β_{11} for μ_k and all predictor parameters for σ are the same for all K components, but the predictor intercept parameter β_{10k} for μ_k depends on k, for $k = 1, 2, \ldots, K$.

Example 3, Mixture of K BCT models: Let $f_k(y) = BCT(\mu_k, \sigma_k, \nu_k, \tau_k)$ for $k = 1, 2, \ldots, K$, where $\log \mu_k = \beta_{1ok} + \beta_{11k} x$, $\log \sigma_k = \beta_{2ok} + \beta_{21k} x$, $\nu_k = \nu = \beta_{3o}$ and $\log \tau_k = \log \tau = \beta_{4o}$. Here predictor parameters β_{1ok} and β_{11k} for μ and β_{20k} and β_{21k} for σ depend on k for $k = 1, 2, \ldots, K$, but parameters β_{3o} for ν and β_{4o} for τ are the same for all k components.

7.5.1 Maximizing the likelihood using the EM algorithm

As in Section 7.2.3 the complete log likelihood is given by (7.11), and can be maximized using an EM algorithm. The M step of the EM algorithm is achieved by expanding the data set K times as in Table 7.1. This method is identical to the method used in Aitkin *et al.* (2006) but here we are not restricting ourselves to the exponential family. The column headed $\hat{\mathbf{w}}^{(r+1)}$ are the iterative weights (calculated internaly) at the $(r+1)^{th}$ iteration. The column headed as `MASS` identifies the K mixture components. This column is declared as a factor in the R implementation of the EM algorithm. If this factor `MASS` is included in the predictor for a distribution parameter μ, σ, ν, or τ, then the predictor intercepts differs between the K components. If an interaction between this factor `MASS` and an explanatory variable x is included in the predictor model for a distribution parameter, then the coefficient of x differ between the K components. Note however that the syntax used in `gamlssNP()` for the interaction between `MASS` and x in the predictor for μ is achieved using the `random=~x` argument (see Section 7.7 for an example).

i	MASS	$\boldsymbol{y}_e$	$\mathbf{X}_e$	$\hat{\mathbf{w}}^{(r+1)}$
1	1			
2	1	$\mathbf{y}$	$\mathbf{X}$	$\hat{\mathbf{w}}_1^{(r+1)}$
⋮	⋮			
n	1			
1	2			
2	2	$\mathbf{y}$	$\mathbf{X}$	$\hat{\mathbf{w}}_2^{(r+1)}$
⋮	⋮			
n	2			
⋮	⋮	⋮	⋮	⋮
1	K			
2	K	$\mathbf{y}$	$\mathbf{X}$	$\hat{\mathbf{w}}_K^{(r+1)}$
⋮	⋮			
n	K			

Table 7.1: Table showing the expansion of data use in M-step of the EM algorithm for fitting the common parameter mixture model

7.6 The gamlssNP() function

The function to fit finite mixtures with parameters in common is `gamlssNP()`. The `gamlssNP()` was initial designed for fitting marginal likelihoods for random effect models. In the next section we give an example of how it can be used to fit finite mixtures models. The function `gamlssNP()` has the following arguments:

formula This argument should be a formula defining the response variable and explanatory he fixed effects terms for the μ parameter of the model. Note that modelling the rest of the distribution parameters can be done by using the usual formulae, e.g. `sigma.fo= x`, which passes the arguments to `gamlss()`

random This should be a formula defining the random part of the model (for random effect models). This formula is also used for fixed effect mixture models to define interactions of the factor `MASS` with explanatory variables x in the predictor for μ (needed to request different coefficients in x in the predictor of μ for the K components).

family A gamlss family distribution.

data This should be a data frame. Note that this argument is mandatory for this function even if the data are attached. This is because the data frame is used to expand the data as in Table 7.1.

K Declaring the number of mixture components (in fixed effects finite mixture models), or the number of mass points or integration quadrature points (for random effects models)

mixture Defining the mixing distribution, "np" for non-parametric finite mixtures or "gq" for Gaussian quadrature.

tol This defines the tolerance scalar usually between zero and one, used for changing the starting

values.

weights For prior weights

control This sets the control parameters for the EM iterations algorithm. The default setting is the `NP.control` function.

g.control This is for controlling the gamlss control function, `gamlss.control`, passed to the gamlss fit

... For extra arguments

7.7 Examples using the gamlssNP() function

7.7.1 The animal brain data

Data summary: the animal brain data

R data file: `brains` in package **gamlss.mx** of dimensions 28×2 (identical to `Animals` in package (MASS))

variables

`brain` : brain weight in g.

`body` : body weight in kg.

purpose: To fit a finite mixture model with different intercepts.

conclusion: A three component normal distribution mixture is found to be adequate

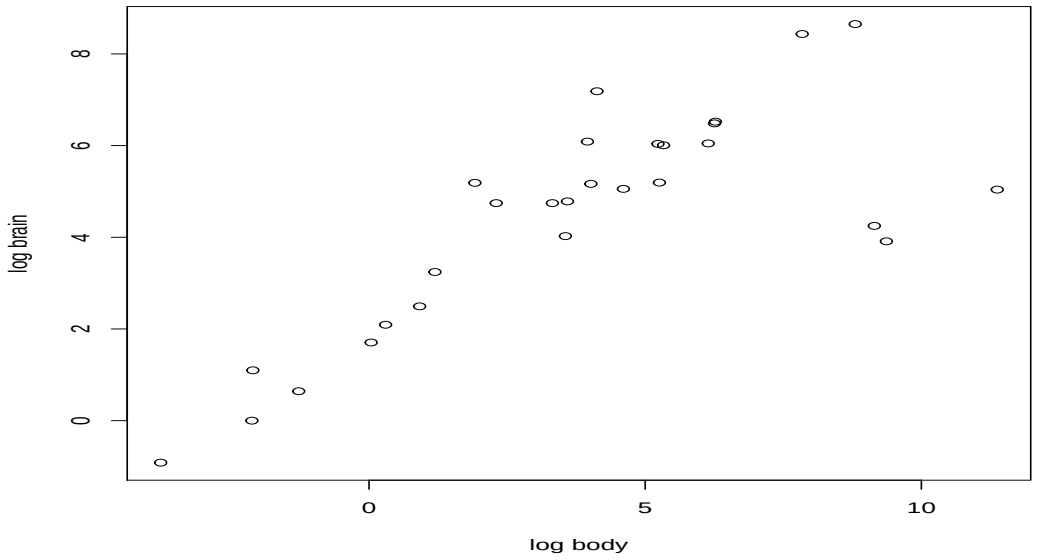
The brain size (`brain`) and the body weight (`body`) were recorded for 28 different animals. Since the distribution of both brain size and body weight are highly skewed a log transformation was applied to each variable to give transformed variables `lbrain` and `lbody`. The resulting data are plotted in Figure 7.7.

```
library(gamlss.mx)
data(brains)
brains$lbrain <- log(brains$brain)
brains$lbody <- log(brains$body)
```

Figure 7.7

```
with(brains, plot(lbrain ~ lbody, ylab = "log brain", xlab = "log body"))
```

A normal error linear regression model of `lbrain` against `lbody` has a highly significant slope for `lbody` but it is believed that the data may represent different stages of evolution and so a mixture models is fitted to the data. In the mixture model, the evolution stage was represented by a shift in the intercept of the regression equation. Normal mixture models with K equal to $1, 2, 3, 4$ are fitted below. Models `br.2, br.3` and `br.4` are models with different intercepts for the K components, where $K = 2, 3$ and 4 respectively. Slopes are the same for the K components, so parallel lines are fitted (see later for how different slopes can be incorporated in the model). The plots of the EM trajectories are omitted here.



R code on page 170

Figure 7.7: A plot of the brain size data

```
br.1 <- gamlss(lbrain ~ lbody, data = brains)

## GAMLSS-RS iteration 1: Global Deviance = 101.2578
## GAMLSS-RS iteration 2: Global Deviance = 101.2578

br.2 <- gamlssNP(formula = lbrain ~ lbody, mixture = "np", K = 2,
     tol = 1, data = brains, family = NO)

## 1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 ..10 ..11 ..12 ..13 ..14 ..15 ..16 ..
## 17 ..18 ..19 ..20 ..21 ..22 ..23 ..24 ..25 ..26 ..27 ..28 ..29 ..30 ..31 ..32 ..33 ..
##
## EM algorithm met convergence criteria at iteration   33
## Global deviance trend plotted.
## EM Trajectories plotted.

br.3 <- gamlssNP(formula = lbrain ~ lbody, mixture = "np", K = 3,
     tol = 1, data = brains, family = NO)

## 1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 ..10 ..11 ..12 ..13 ..14 ..
## EM algorithm met convergence criteria at iteration   14
## Global deviance trend plotted.
## EM Trajectories plotted.

br.4 <- gamlssNP(formula = lbrain ~ lbody, mixture = "np", K = 4,
     tol = 1, data = brains, family = NO)
```

```
## 1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 ..10 ..11 ..12 ..13 ..14 ..15 ..16 ..
## 17 ..18 ..19 ..20 ..21 ..22 ..23 ..24 ..25 ..26 ..27 ..28 ..29 ..
## EM algorithm met convergence criteria at iteration   29
## Global deviance trend plotted.
## EM Trajectories plotted.
```

We compare the models using each of the ctiteria AIC and SBC:

```
GAIC(br.1, br.2, br.3, br.4)
```

```
##      df       AIC
## br.3  7  79.15079
## br.4  9  83.15613
## br.2  5  85.95938
## br.1  3 107.25779
```

```
GAIC(br.1, br.2, br.3, br.4, k = log(length(brains$body)))
```

```
##      df       AIC
## br.3  7  88.47622
## br.2  5  92.62040
## br.4  9  95.14598
## br.1  3 111.25440
```

Changing the starting values by trying different values for `tol` (e.g. trying each of the values $0.1, 0.2, \ldots, 1$ in turn), for models `br.2`, `br.3` and `br.4`, did not change the values of AIC and SBC given by the two GAIC commands above. The model `br.3` with three components (i.e. three parallel lines) is selected by both AIC and SBC criteria. We now print model `br.3` and its estimated (fitted) posterior probabilities.

```
br.3
```

```
##
## Mixing Family:  c("NO Mixture with NP", "Normal Mixture with NP")
##
## Fitting method: EM algorithm
##
## Call:  gamlssNP(formula = lbrain ~ lbody, family = NO, data = brains,
##     K = 3, mixture = "np", tol = 1)
##
## Mu Coefficients :
## (Intercept)        lbody        MASS2        MASS3
##     -3.0715       0.7499       4.9805       6.5530
## Sigma Coefficients :
## (Intercept)
##     -0.9387
##
## Estimated probabilities: 0.1071429 0.7514161 0.141441
##
## Degrees of Freedom for the fit: 7 Residual Deg. of Freedom   21
## Global Deviance:     65.1508
##             AIC:     79.1508
```

```
##             SBC:     88.4762

br.3$post.prob

## [[1]]
##       [,1]         [,2]         [,3]
##  [1,]    0 9.999624e-01 3.760045e-05
##  [2,]    0 9.999995e-01 4.736429e-07
##  [3,]    0 9.996309e-01 3.691210e-04
##  [4,]    0 9.979683e-01 2.031733e-03
##  [5,]    0 9.999947e-01 5.254125e-06
##  [6,]    1 0.000000e+00 0.000000e+00
##  [7,]    0 9.583487e-01 4.165135e-02
##  [8,]    0 9.995208e-01 4.792198e-04
##  [9,]    0 9.999824e-01 1.764759e-05
## [10,]    0 1.617020e-01 8.382980e-01
## [11,]    0 9.947820e-01 5.217995e-03
## [12,]    0 9.999769e-01 2.306099e-05
## [13,]    0 9.998409e-01 1.590788e-04
## [14,]    0 3.157024e-06 9.999968e-01
## [15,]    0 9.997563e-01 2.436742e-04
## [16,]    1 0.000000e+00 0.000000e+00
## [17,]    0 1.044992e-04 9.998955e-01
## [18,]    0 9.999998e-01 2.035525e-07
## [19,]    0 9.999978e-01 2.187091e-06
## [20,]    0 9.999398e-01 6.024621e-05
## [21,]    0 9.999799e-01 2.013594e-05
## [22,]    0 9.992899e-01 7.101261e-04
## [23,]    0 9.999975e-01 2.489188e-06
## [24,]    0 6.263055e-02 9.373694e-01
## [25,]    1 0.000000e+00 0.000000e+00
## [26,]    0 9.999977e-01 2.336595e-06
## [27,]    0 8.662450e-01 1.337550e-01
## [28,]    0 9.999999e-01 6.645917e-08
```

So model `br.3` can be presented as $Y \sim NO(\hat{\mu}, \hat{\sigma})$ where

$$\hat{\mu} = \begin{cases} -3.072 + 0.750x, & \text{with probability } 0.107 \\ 1.909 + 0.750x, & \text{with probability } 0.751 \\ 3.481 + 0.750x, & \text{with probability } 0.141 \end{cases} \tag{7.13}$$

and $\hat{\sigma} = 0.391$. [Note that the intercept for the second component in (7.13) is obtained from the estimated parameter coefficients for μ by $1.909 = -3.072 + 4.981$, since `MASS2` gives the adjustment to the intercept for the second mixture component; similarly for `MASS3`.] The output given by `br.3$post.prob` contains the estimated posterior probabilities of each of the observations in the data set belonging to each of the 3 components. These are the fitted weights $\hat{w}_{ik}$ given by (**??**) on convergence of the EM algorithm. A plot of the data together with the fitted values for the μ parameter of model `br.3` are shown in Figure 7.8. Each observation of the data was allocated to the component for which it had the highest posterior probability and the observations are plotted in the command below with circles (colour red), squares (colour

green) and diamonds (colour blue) representing allocation to each of the 3 components. Note that since the parameter μ in this (normal distribution) case is the mean of the distribution the lines are the fitted means of the conditional distributions $f_k(y)$ for $k = 1, 2, 3$. Figure 7.8 is obtained by :

Figure 7.8

```
with(brains, plot(lbody, lbrain,
                  pch = c(21, 22, 23)[max.col(br.3$post.prob[[1]])],
                  bg = c("red", "green3", "blue")[max.col(br.3$post.prob[[1]])]))
for (k in 1:3) {
     with(brains, lines(fitted(br.3, K = k)[order(lbody)] ~ lbody[order(lbody)],
         lty = k))
}
```

R code on page 174

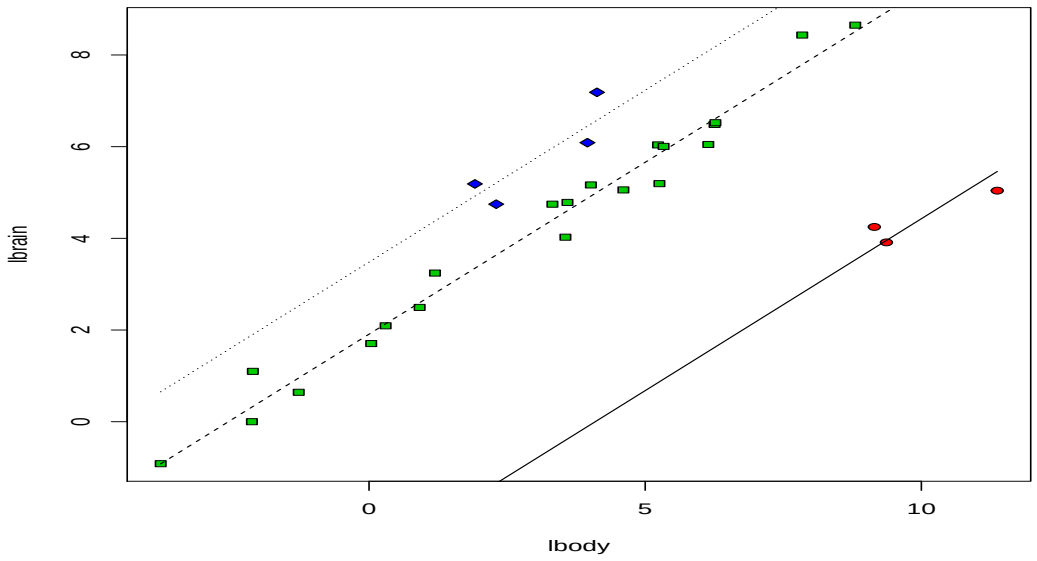


Figure 7.8: A plot of the brain size data together with a plot of the three component fitted means of log brain size (`lbrain`) against log body size (`lbody`), (solid, dashed and dotted for component 1,2 and 3 respectively)

The weighted average for the (conditional) parameters $\hat{\mu}$ for the $K(= 3)$ components for each observation, i.e. $\sum_{k=1}^{K} \hat{\pi}_k \hat{\mu}_{ik}$ can be obtained using the command `fitted(br.3, K=0)`. Since the parameter μ is, in this case, the mean of the normal distribution, this gives the marginal mean of the response variable `lbrain` given the explanatory variable `lbody`.

Note how the marginal mean, using the function `fitted()`, is obtained here compared to the conditional means. If the argument `K` of the `fitted()` function has any value in the range $1, 2, 3$, (that is the range of permissible values for the model `br.3`), then the conditional parameters is given. For any other value the average μ is given. This will be the marginal mean only if parameter μ is the mean of the conditional distribution for each component.

model	μ intercept	μ slope	σ
br.3	different	same	same
br.31	different	same	different
br.32	different	different	same
br.33	different	different	diferent

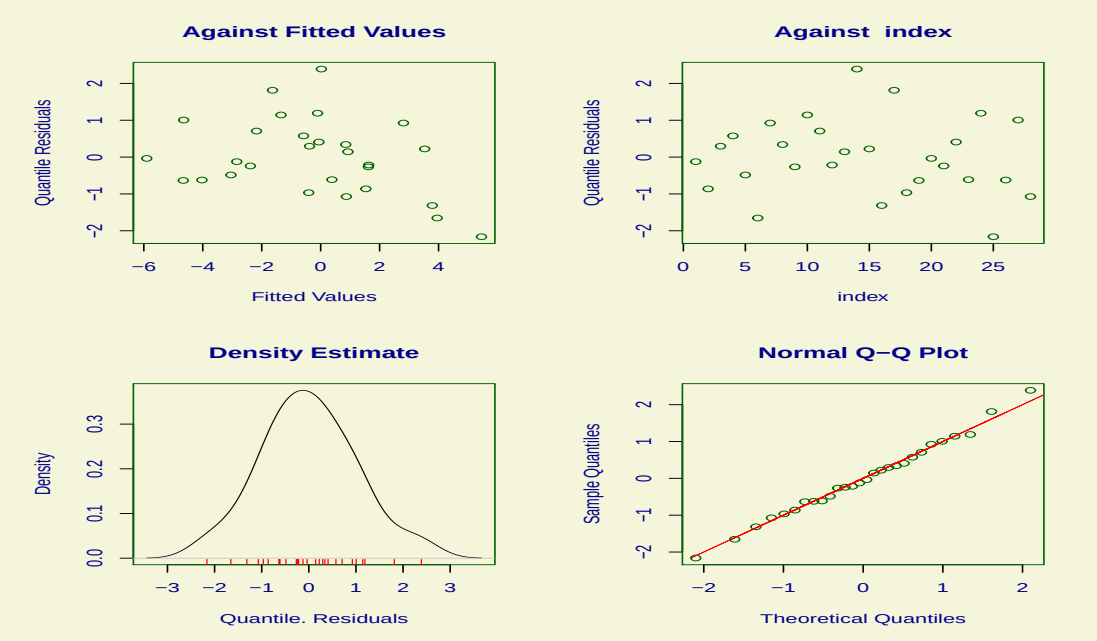
Table 7.2: Possible alternative models for the animal brain data

A residual plot of the finite mixture model is obtained the usual way using the function `plot()`.

```
plot(br.3)
```

```
## ******************************************************************
##   Summary of the Randomised Quantile Residuals
##                            mean   =  -0.004003875
##                        variance   =  1.052469
##                coef. of skewness  =  0.1668313
##                coef. of kurtosis  =  2.739025
## Filliben correlation coefficient  =  0.9962244
## ******************************************************************
```

Figure 7.9



R code on page 175

Figure 7.9: The residual plot of model `br.3` for the animal brain size data

There are several different models that we could fit here depending on which parameters are common to the $K = 3$ components in the model. Table 7.2 shows possible alternative models and the code below shows how to fit them:

```
br.31 <- gamlssNP(formula = lbrain ~ lbody, sigma.fo = ~MASS,
     mixture = "np", K = 3, tol = 1, data = brains, family = NO)
```

```
## 1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 ..10 ..11 ..12 ..13 ..14 ..15 ..16 ..
## 17 ..18 ..19 ..20 ..21 ..22 ..23 ..24 ..25 ..26 ..27 ..28 ..
## EM algorithm met convergence criteria at iteration   28
## Global deviance trend plotted.
## EM Trajectories plotted.
```

```
br.32 <- gamlssNP(formula = lbrain ~ lbody, random = ~lbody,
     sigma.fo = ~1, mixture = "np", K = 3, tol = 1, data = brains,
     family = NO)
```

```
## 1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 ..10 ..11 ..12 ..13 ..14 ..15 ..16 ..
##
## EM algorithm met convergence criteria at iteration   16
## Global deviance trend plotted.
## EM Trajectories plotted.
```

```
br.33 <- gamlssNP(formula = lbrain ~ lbody, random = ~lbody,
     sigma.fo = ~MASS, mixture = "np", K = 3, tol = 1, data = brains,
     family = NO)
```

```
## 1 ..2 ..3 ..4 ..5 ..6 ..7 ..8 ..9 ..10 ..11 ..12 ..13 ..14 ..15 ..16 ..
## 17 ..
## EM algorithm met convergence criteria at iteration   17
## Global deviance trend plotted.
## EM Trajectories plotted.
```

We compare the models using each of the criteria AIC and SBC:

```
GAIC(br.3, br.31, br.32, br.33)
```

```
##       df      AIC
## br.32  9 77.31133
## br.3   7 79.15079
## br.33 11 80.26824
## br.31  9 81.93037
```

```
GAIC(br.3, br.31, br.32, br.33, k = log(length(brains$lbody)))
```

```
##       df      AIC
## br.3   7 88.47622
## br.32  9 89.30117
## br.31  9 93.92021
## br.33 11 94.92249
```

Model `br.3` has the smallest SBC. [Note model `br.32` has the smallest AIC, however with so many parameters in the model and so few data points it is not sensible to try to interpreted this model.] Note also that since model `br.33` has components with no parameters in common it could also be fitted using the `gamlssMX` function.

Part IV

Additive terms

Chapter 8

Linear parametric additive terms

This chapter explains types linear terms which be used within a `gamlss` model and how they can be used. In particularly it explains :

1. linear terms and interactions for factors and numerical explanatory variables.
2. different useful bases used for explanatory variables.

This chapter is essential for understanding the different types of additive terms in GAMLSS.

8.1 Introduction to linear and additive terms

In the GAMLSS implementation in **R**, the function `gamlss()` in **gamlss** allows modelling all the distribution parameters μ, σ, ν and τ as linear and/or non-linear and/or 'non-parametric' smoothing functions of the explanatory variables. This allow the explanatory variables to affect the predictors, (the η's), of the specific parameters and therefore the parameters themselves. As a result the shape of the distribution of the response variable, (not only the mean), is affected by the explanatory variables.

We shall refer to the explanatory variables as *terms* in the model. The relationships between a predictor η and the terms can be *linear* or *non-linear*. A non-linear relationship can be *parametric non-linear* or a *smoother*. As an example of a parametric non-linear relationship consider the expression $\beta_1 x^{\beta_2}$ where both β_1 and β_2 are parameters and have to be estimated within the model. Smoothers are *non-parametric* techniques which allow the data to determine which relationship exists between the predictors and the explanatory variables, see Chapter **??** for more details about additive smoothing terms.

By *additive terms* we refer to the fact that in order to evaluate the effect of the explanatory variables on the predictor for a specific parameter we have to add up their individual effects. Additivity does not imply that there are no interactions in the model. Section 8.2.2 presents some examples.

As an example of what type of terms GAMLSS model can take, let the x's represent continuous explanatory variables and the f a factor (with three levels), then the following could be typical

predictor η for a parameter:

$$\begin{aligned}\eta &= b_0 + b_1 x_1 + b_2 x_2 + b_3 \texttt{if}(f = 2) + b_4 \texttt{if}(f = 3) + b_5 (x_1 \times x_2) + \\ & \quad b_6 x_1 \texttt{if}(f = 2) + b_7 x_1 \texttt{if}(f = 3) + s_1(x_3) + s_2(x_4) + \\ & \quad s_3 b_8(x_1) x_4 + s_4(x_1) \texttt{if}(f = 2) + s_5(x_1) \texttt{if}(f = 3) + s_6(x_3, x_5)\end{aligned}$$

where

b_0	is the constant term
$b_1 x_1 + b_2 x_2$	are linear additive terms of continuous variables
$b_3 \texttt{if}(f = 2) + b_4 \texttt{if}(f = 3)$	is the main effect of a factor `f`
$b_5(x_1 x_2)$	linear interaction between two continuous terms
$b_6 x_1 \texttt{if}(f = 2) + b_6 x_1 \texttt{if}(f = 3)$	is a linear interaction between x_1 and f
$s_1(x_3) + s_2(x_4)$	are smoothing additive terms for x_3 and x_4
$s_3(x_1) x_4$	is varying coefficient term for x_4 given x_1
$s_4(x_1) \texttt{if}(f = 2) + s_5(x_1) \texttt{if}(f = 3)$	is an interaction of f with smoothing terms for x_1
$s_6(x_3, x_5)$	is a smooth iteration between x_3 and x_5

The diagram in **??** shows a classification of the different additive terms within the GAMLSS models.

In this Chapter we are dealing with linear parametric terms. That is, the left part of the above figure. Chapter **??** describes the non-parametric smoothing terms. First we introduce how main effects and interactions for linear terms are used (based on Wilkinson and Rogers [1973] notation) and them how some non-linear relationships can be still modelled using linear basis functions. In particular the following basis functions are introduced:

1. polynomials (Section 8.3)
2. fractional polynomials (Section 8.4)
3. piecewise polynomials (Section 8.5)
4. B-splines (Section 8.6)

The fitting of non-linear parametric functions with GAMLSS is described in Chapter **??**. Section 8.2 describes how simple linear terms for continuous and categorical explanatory variables are accommodated with an additive model. It also describes linear interactions between terms.

8.2 Linear terms

The linear part is declared by the use of formulae. A formula in **R** looks something like:

$$y \sim x1 + x2 + f1 + f2 * x3$$

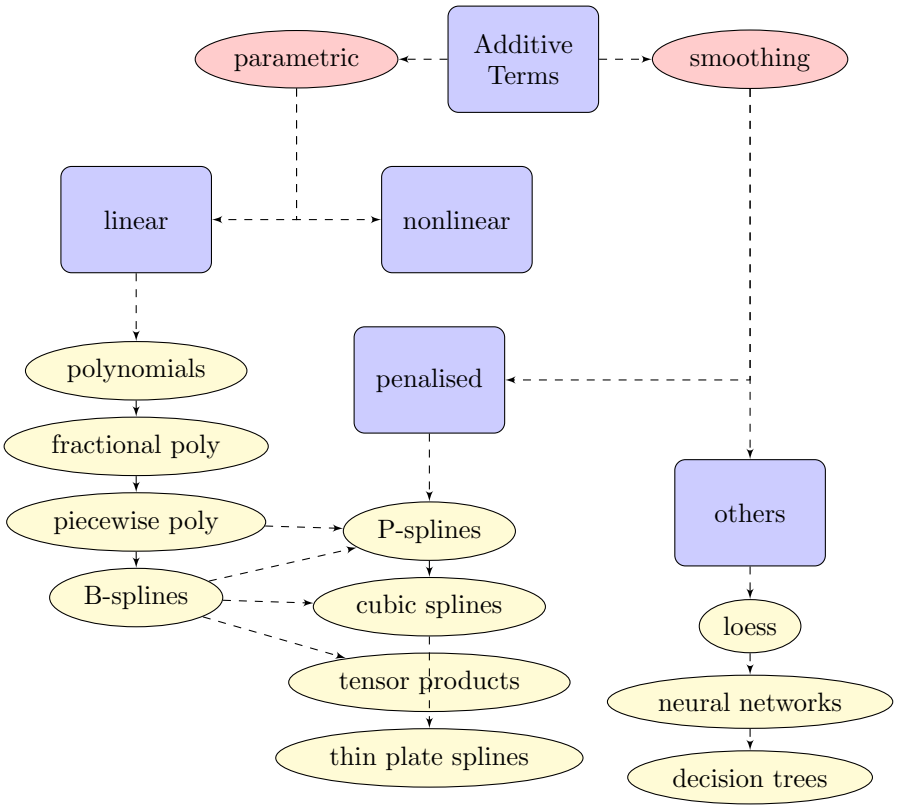


Figure 8.1: Diagram showing the different additve terms in GAMLSS

where the variable y on the left side of $\sim$ is the response variable, and the variables $x1$, $x2$, $x3$, $f1$, $f2$ on the right side of $\sim$ are the explanatory variables. For demonstration purposes we shall use here the x's to denote *continuous* explanatory variables and f's for *factors* (that is, *categorical* explanatory variables).

The symbols '+' and '$*$' have special meaning here derived from the Wilkinson and Rogers [1973] notation as applied in the `S` language by Chambers and Hastie [1992]. [It is the same notation used in **R** the fitting of linear models, `lm()`, and generalised linear models, `glm()`, see for example Venables and Ripley [2002], Section 6.2.] The purpose of the right hand part of the formula is to create the design matrix $\mathbf{X}$ for fitting the linear part of the GAMLSS model. The symbol '+' is describes *additive* terms while the symbol '$*$' *interactions* between terms.

8.2.1 Additive linear terms

If the explanatory variable is a continuous one the '+' sign will enter a single column in the design matrix $\mathbf{X}$. If it is a factor it will enter a set of dummy variables. This is the most common way to enter factors in the design matrix $\mathbf{X}$. A *dummy* variable is a vector containing zeros and ones. As an example, if the factor $f1$ has say 4 levels (that is, four categories) then $f1$ will be represented within X as a set of four dummy variable. Each dummy variable will have the values 1 if $f1$ is at the appropriate level and zero otherwise (see example below).

Things are complicated a bit with the presence of the constant in the design matrix $\mathbf{X}$. The constant or *intercept* is represented in the design matrix as a column of ones. The constant is automatically included in the design matrix unless the user uses within the formula '`-1`'. The problem arises from the fact that by including both the constant and the factor (as a set of dummy variables) in the model, the design matrix became singular (and therefore not invertible). To avoid this **R** drops the first dummy variable that is the first level of the factor from the design matrix.

To demonstrate we use the `aids` data. The data were collected quarterly and the `aids` `data.frame` contains three variables: i) `y`: the number of quarterly aids cases in England and Wales from January 1983 to March 1994 . ii) `x`: time in quarters from January 1983 iii) `qrt`: a factor for the quarterly seasonal effect. Here we input the data and output the first ten rows of the design matrix of the model $\sim x + qrt$.

```
data(aids)
head(with(aids,model.matrix(formula(~x+qrt))), 10)
```

```
##    (Intercept)  x qrt2 qrt3 qrt4
## 1            1  1    0    0    0
## 2            1  2    1    0    0
## 3            1  3    0    1    0
## 4            1  4    0    0    1
## 5            1  5    0    0    0
## 6            1  6    1    0    0
## 7            1  7    0    1    0
## 8            1  8    0    0    1
## 9            1  9    0    0    0
## 10           1 10    1    0    0
```

8.2.2 Linear interactions

Interactions make the effect of two or more explanatory variables a joint effect. Interactions can be between:

- two or more continuous variables
- two or more factors
- one or more continuous variables and one or more factors

The additive formula ~`x1+x2` for two continuous variables initiates a linear plane fitting (by introducing two new columns on the design matrix $\mathbf{X}$). The interaction formula ~`x1*x2` introduces a third column containing the element-wise multiplication of `x1` by `x2`. This extra column makes the fitting surface a curvy one. This linear interaction surface is fitted globally and it is rather restrictive compared with surfaces fitted by non-parametric smoothers where more flexibility locally is allowed. The formula ~`x1*x2` is equivalent in **R** to ~`x1+x2+x1:x2` which represents the main effect for `x1`, the main effect of `x2` and the linear interaction between `x1` and `x2`. We refer to the coefficient of `x1` as *the main effectt* for `x1` the coefficient of `x2` as the main effect for `x2` and the coefficient of `x1:x2` as the *interaction* of `x1` and `x2`. Depending on how many variables are involved we have 'two way', i.e. `x1*x2`, 'three way' i.e. `x1*x2*x3` and up to say 'k way' forms of interactions.

Interactions for categorical variables say ~`f1*f2` add extra columns in the design matrix to make sure that all the combinations of the crossing of the different levels involved are represented. For example if `f1` has 2 levels, $\{A, B\}$, and `f2` has 3 levels, $\{1, 2, 3\}$, them the iteration will have $6 = 2 \times 3$ levels reflecting all combinations $\{A1, A2, A3, B1, B2.B3\}$. The following is an example of how **R** works out the design matrix for factor iterations. First we create two factors with 2 and 3 levels of length 24 and then we show the first twelve rows of the design matrix.

```
f1<-gl(2,1, 24)
levels(f1) <- c("A", "B")
f1
```

```
##  [1] A B A B A B A B A B A B A B A B A B A B A B A B
## Levels: A B
```

```
f2<-gl(3,2, 24)
f2
```

```
##  [1] 1 1 2 2 3 3 1 1 2 2 3 3 1 1 2 2 3 3 1 1 2 2 3 3
## Levels: 1 2 3
```

```
head(model.matrix(~f1*f2), 12)
```

```
##    (Intercept) f1B f22 f23 f1B:f22 f1B:f23
## 1            1   0   0   0       0       0
## 2            1   1   0   0       0       0
## 3            1   0   1   0       0       0
## 4            1   1   1   0       1       0
## 5            1   0   0   1       0       0
## 6            1   1   0   1       0       1
## 7            1   0   0   0       0       0
## 8            1   1   0   0       0       0
```

```
## 9           1   0   1   0       0       0
## 10          1   1   1   0       1       0
## 11          1   0   0   1       0       0
## 12          1   1   0   1       0       1
```

Note that, since the constant (intercept) is added automatically in the design matrix, the main effect of `f1` is represented by 1 column, (number of levels of `f1` minus 1) headed by `f1B`, the main effect of `f2` by 2 columns, (3-1=2), and the interaction of the two factors by $2 = (2-1) \times (3-1)$ columns headed by `f1B:f22` and `f1B:f23`. In general if `f1` has k_1 levels and `f2` has k_2 levels then the interaction has $k = (k_1 - 1) \times (k_2 - 1)$ columns.

The following is an example of an interaction between a continuous variable and a factor.

```
data(aids)
head(with(aids,model.matrix(formula(~x*qrt))), 10)
```

```
##    (Intercept)  x qrt2 qrt3 qrt4 x:qrt2 x:qrt3 x:qrt4
## 1            1  1    0    0    0      0      0      0
## 2            1  2    1    0    0      2      0      0
## 3            1  3    0    1    0      0      3      0
## 4            1  4    0    0    1      0      0      4
## 5            1  5    0    0    0      0      0      0
## 6            1  6    1    0    0      6      0      0
## 7            1  7    0    1    0      0      7      0
## 8            1  8    0    0    1      0      0      8
## 9            1  9    0    0    0      0      0      0
## 10           1 10    1    0    0     10      0      0
```

The continuous variable here is time which is coded as the values from 1 to 45. The design matrix in this case contains: i) the constant as the first column ii) the `x` main effect as the second column iii) the main effect of the factor `qrt` as three dummy variables containing the last three levels of the factor and iv) the interaction `x:qtr` represented here by the last three columns. In order to show the interpretation of the model containing both factors and continuous variable consider the simple analysis of covariance case, ANOCOVA, with a single covariate `x` and a single factor `f`. In the standard ANOCOVA table we assume that the response variable is normally distributed and we interested to see how the mean of the response variable is behaving. Within GAMLSS we are interested to see how the predictor of a distribution parameter η changes with the continuous variable `x` and the factor `f`. In a case like this there are five different models of interest for η:

1. The *null* model in which neither `f` or `x` are needed in the model, i.e. just the constant
2. The *simple analysis of variance* model where only the factor `f` is included in the model but not the variable `x`
3. The *simple regression model* where the variable `x` is included in the model but not the factor `f`
4. The *additive model* where both `x` and `f` are included in the model, but with no interaction between them, so the slope for `x` is common for all levels of `f` but the intercept varies according to to the levels in `f`
5. The *interaction model* where both intercepts and slopes vary for different levels of `f`

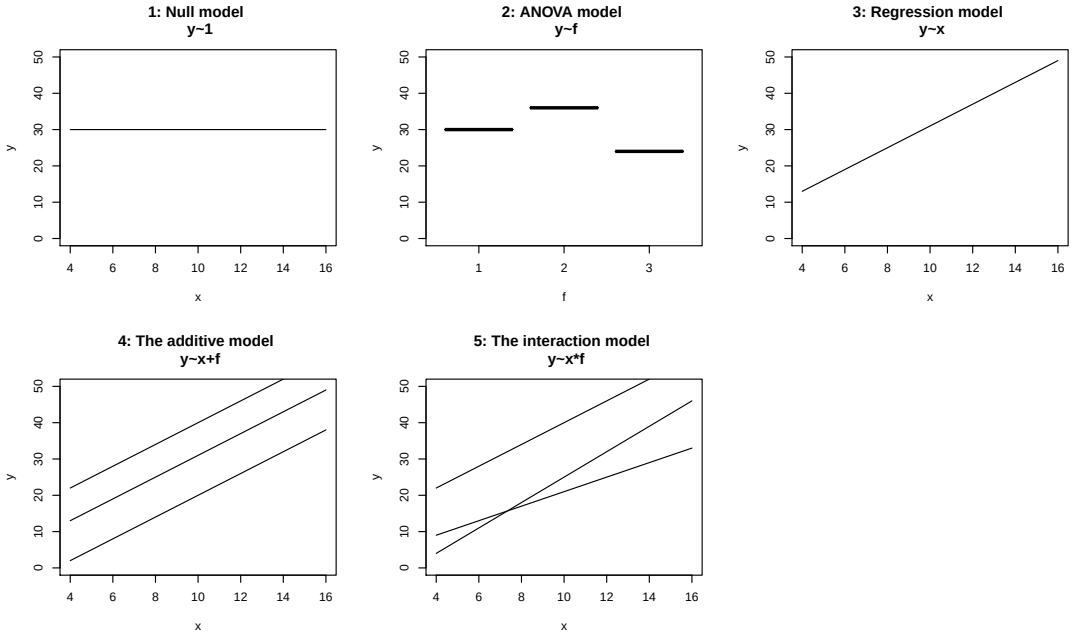


Figure 8.2: The five different models in the simple analysis of covariance

A graph for the different models involved is shown in Figure 8.2. The vertical axis labelled y represents the mean of y in a normal error ANOCVA model, or more generally the predictor η for a distribution parameter in a GAMLSS model, in which case the lines in the graph represent the different predictor values for each of the models. Model 1 has a constant predictor value. Model 2 has different values for the different levels of the factor `f`. The predictor of model 3 increases linearly with `x`, while the predictor of model 4 shows the same slope in the linear relationship between the the predictor and `x` but with different intercept values. In model 5, both the intercepts and slopes of the predictor vary according to the levels of the factor `f`.

In the normal case where we model the mean, the choice between models is achieved using an ANOCOVA table and the *nested* models can be compared using an F-test. (Model I is nested within model II if model I is a subclass of model II). For example Model 4 above is a nested model within model 5 so we can compare them. Models 2 and 3 are nested models within both models 4 and 5, while the null model is nested within all the rest. Note model 2 is not nested within model 3. The appropriateness of the F-distribution comes as a result of the normal assumption with constant variance for the response variable, so in general it not appropriate for a GAMLSS model where the selection between models can be achieved through χ^2_{df} asymptotic distribution of the generalized likelihood ratio test statistic for nested models (where df is the deference in degrees of freedom between models) or GAIC for non-nested ones.

8.3 Polynomials

Polynomial are the simplest way of trying to model non-linear relationships in a regression type situation. A polynomial has the form:

$$h(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \ldots + \beta_p x^p \tag{8.1}$$

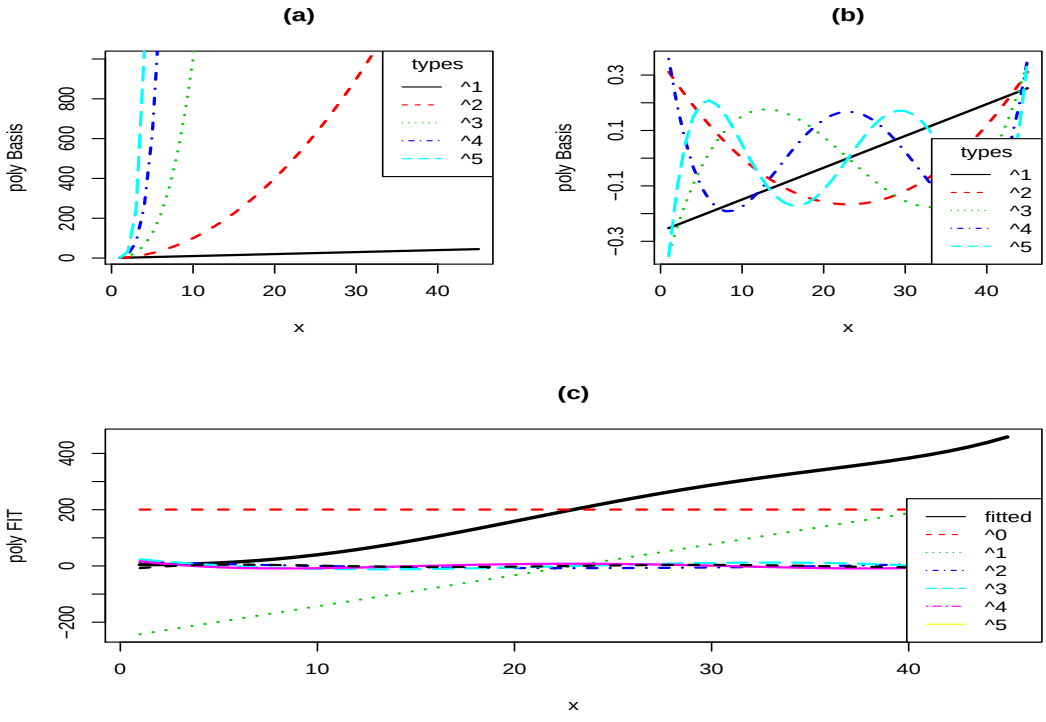
and the only thing required by the user is the add the appropriate columns in the design matrix $\mathbf{X}$. There are two ways of doing this in **R** the first is by using the function `I()` the second through the orthogonal polynomial function `poly()`. The function `I()` allows the user to calculate expression within a formula. For example $\sim$ `x + I(x^2)+I(x^3)` will fit a cubic polynomial for `x` by creating two extra column (apart from the columns of the constant and `x`) containing x^2 and x^3. The columns of the design matrix $\mathbf{X}$ form a polynomial basis. Higher order polynomial can be added the same way. The problem with defining polynomials this way is that it can lead to numerical inaccuracy. See for example Figure 8.3(a) where the basis functions for a 5^{th} degree polynomial for the time variable (1 to 45) of the `aids` data is plotted. The values for x^p can easily become too big (as in Figure 8.3(a)) or too small (for small values of `x`). This problem is avoided if, instead of the standard basis, we use an orthogonal polynomial basis. Figure 8.3(b) shows an orthogonal polynomial basis for the same time variable of the `aids` data. The `poly()` function provides this facility in **R**. For example, Figure 8.3(b) shows all up to 5th order orthogonal polynomial basis (apart from the constant). The curves are easily identified as linear, quadratic, cubic etc. Furthermore the basis vectors in the design matrix $\mathbf{X}$ are orthogonal from each other.

The fitted values of the same order polynomial, irrespective of whether the standard or orthogonal basis is used, should be identical (unless something went numerically wrong). The fitted values are a linear function of the basis variables, weighted differently, according to the fitted coefficients. Figure 8.3(c) shows the fitted values of the the model `y~poly(x,5)` to the `aids` data. The figure also show the orthogonal basis functions weight by their fitted coefficients. Adding the weighted basis functions together will results the fitted values. Note that in this specific case the constant (flat line) and the linear part of the basis play a major role in determining the shape of the fitted value, while the rest of the basis only add a small curvature to it.

Figure 8.3

```
data(aids)
def.par <- par(no.readonly = TRUE)
# simple
X <- with(aids,model.matrix(formula(~x+I(x^2)+I(x^3)+I(x^4)+I(x^5))))[,-1]
nf <- layout(matrix(c(1,2,3,3),2,2,byrow=TRUE))
matplot(X, type="l", ylim=c(9, 1000), lwd=2, ylab="poly Basis", xlab="x",
              main="(a)")
legend("topright", legend =c("^1", "^2", "^3", "^4", "^5"),
       col=c(1,2,3,4,5), lty=c(1,2,3,4,5),
       ncol=1, bg="white", title="types")
# orthogonal
P <- with(aids,model.matrix(formula(~poly(x, 5))))[,-1])
matplot( P, type="l", lwd=2, ylab="poly Basis", xlab="x", main="(b)")
legend("bottomright", legend =c("^1", "^2", "^3", "^4", "^5"),
       col=c(1,2,3,4,5), lty=c(1,2,3,4,5),
       ncol=1, bg="white", title="types")
```

```
# fitting the model
m1 <- gamlss(y~poly(x, 5), data=aids)

## GAMLSS-RS iteration 1: Global Deviance = 430.3
## GAMLSS-RS iteration 2: Global Deviance = 430.3

P1 <- model.matrix(with(aids,formula(~poly(x, 5))))
b <- coef(m1)
b

## (Intercept) poly(x, 5)1 poly(x, 5)2 poly(x, 5)3 poly(x, 5)4 poly(x, 5)5
##      200.49      961.16       46.19      -68.04       46.24       22.13

F <- t(rep(b,45) *t(P1))
Fit <- cbind(fitted(m1), F)
matplot(Fit, type="l", lwd=c(3,2,2,2,2,2,2), ylab="poly FIT",
        xlab="x", main="(c)")
legend("bottomright", legend =c("fitted", "^0", "^1", "^2", "^3", "^4", "^5"),
         col=c(1,2,3,4,5,6,7), lty=c(1,2,3,4,5,6,7), ncol=1, bg="white")
par(def.par)
```



R code on page 186

Figure 8.3: Polynomial for aids data: (a) standard polynomials basis, (b) orthogonal polynomial basis, (c) the fitted values are a linear function of the basis vectors i.e. $\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}}$.

While orthogonal polynomials are more stable to fit they suffer from the problem of interpretability. Occasionally we may have to go back to the standard polynomial in order to explain the influence of the explanatory variable on the predictor.

The problem with polynomials in general is that, because their basis is defined globally in the full range of values for x, they can be highly influenced by a few observations in the data. This is a well known phenomenon and it is avoided by using smooth non-parametric functions. The Section 8.8.1 provides an example of the use of orthogonal polynomials.

8.4 Fractional Polynomials

A polynomial, βx^p, is called *fractional* if the power p is not necessarily a positive integer, e.g. $\beta x^{-1/2}$. Fractional polynomials were introduced by Royston and Altman [1994]. The idea is that with only a few fractional polynomials you can get a very flexible base to fit a parametric curve to your data. The GAMLSS implementation uses the functions `fp()` and `bfp()` which are loosely based on the fractional polynomial function `fracpoly()` for `S-PLUS` given by Ambler [1999]. The function `bfp()` generates the design matrix for fitting a fractional polynomial, while the function `fp()` works in `gamlss()` as an additive 'smoother' term.

The function `fp()` works as follows. Its argument `npoly` determines whether one, two or three terms in the fractional polynomial will be used in the fitting. For example with `npoly=3` the following polynomial functions are fitted $\beta_0 + \beta_1 x^{p_1} + \beta_2 x^{p_2} + \beta_3 x^{p_3}$ where each p_j, for $j = 1, 2, 3$ can take any value within the predetermined set $(-2, -1, -0.5, 0, 0.5, 1, 2, 3)$ with the value 0 interpreted as function $\log(x)$ (rather than x^0). See Figure **??** for the shape of these basis functions. If two powers, p_j's, happen to be identical then the two terms $\beta_{1j} x^{p_j}$ and $\beta_{2j} x^{p_j} \log(x)$ are fitted instead. Similarly if three powers p_j's are identical the terms fitted are $\beta_{1j} x^{p_j}$, $\beta_{2j} x^{p_j} \log(x)$ and $\beta_{3j} x^{p_j} [\log(x)]^2$. Note that `npoly=3` is rather slow since it fits all possible 3-way combinations at each backfitting iteration.

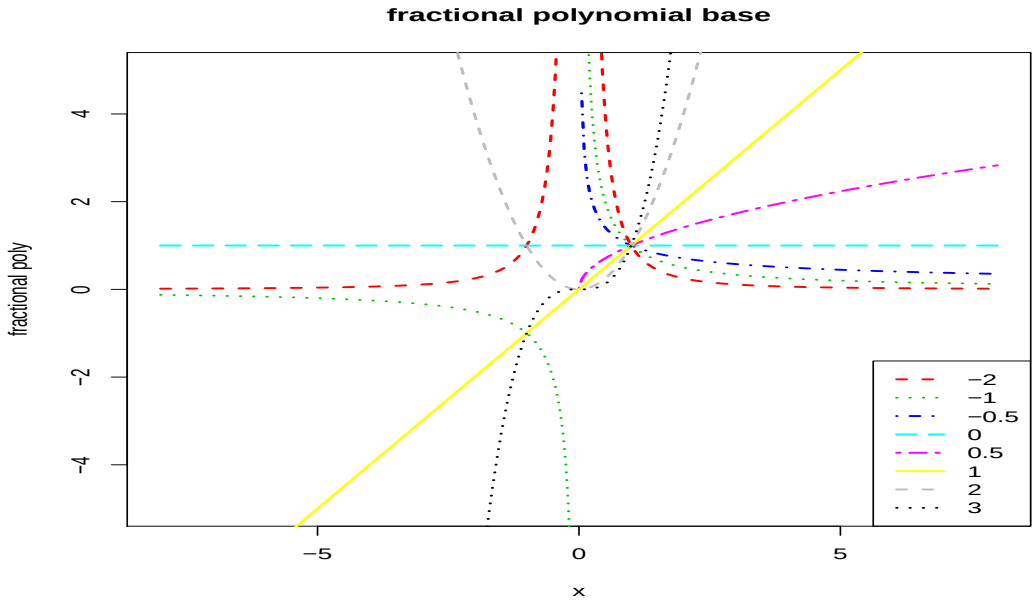
Figure 8.4

```
x<-seq(-8,8,0.05)
plot(x, type="n",xlim=c(min(x), max(x)), ylim=c(-5,5),
      main="fractional polynomial base", xlab="x", ylab="fractional poly")
ii<-1
for (i in c(-2, -1, -0.5, 0, 0.5, 1, 2, 3))
lines( I(x^i)~x, lwd=2, col=(ii<-ii+1), lty=ii)
legend("bottomright", legend=c("-2", "-1", "-0.5", "0", "0.5", "1", "2", "3"),
      col=2:9, lty=2:9, lwd=2 )
```

Fractional polynomials can be fitted within GAMLSS using the additive function `fp()`. It takes as arguments the x variable and `npoly` (the number of fractional polynomial terms) which takes the values 1,2,3. An example of using the function `fp()` within GAMLSS is shown in Section 8.8.2.

8.5 Piecewise Polynomials and Regression Splines

This section is an introduction to piecewise polynomials. Piecewise polynomials are a useful tool in statistical modelling both on their own or in their penalised form. In fact fitting penalised



R code on page 188

Figure 8.4: Showing the fractional polynomial basis used within GAMLSS that is polynomials with power $(-2, -1, -0.5, 0, 0.5, 1, 2, 3)$ where 0 corresponds to a log function.

piecewise polynomials is the most popular way of non-parametric smoothing due to the fact that it works well in practice and is easy to implement. see Chapter ??

Sometimes we are confronted with data in which there is a change in the relationship between the dependent and independent variables. For simplicity, we will only discuss the case where there is only one explanatory variable x. This type of data can be modelled using piecewise polynomials in x to describe the relationship. The value(s) of the explanatory variable where the piecewise polynomials change are called *breakpoints* or *knots*, and these polynomials are known as splines if continuity restrictions are placed on them at these breakpoints. The name splines itself is derived from thin rods that engineers have used to fit curves through given points. Smith [1979] referred to these piecewise polynomials as regression splines and examined them as a tool in regression.

A simple example of a piecewise polynomial is the split line curve with a single breakpoint. There are two types of split line curve models - continuous and discontinuous split lines. The continuous split line case has the form

$$h(x) = \beta_{00} + \beta_{01}x + \beta_{11}(x - b)H(x > b) \tag{8.2}$$

where $H(x > b)$ is the *Heaviside* function taking value 1 if $x > b$ otherwise 0. β_{00}, β_{01}, and β_{11} are the linear parameters, and b, the breakpoint (or knot), being the non-linear parameter. In a statistical modelling situation all four parameters need to be estimated. Figure 8.5(a) shows a continuous split line curve with parameters $\beta_{00} = 5$, $\beta_{01} = 0.5$, $\beta_{11} = 1$ and $b = 5$. With a

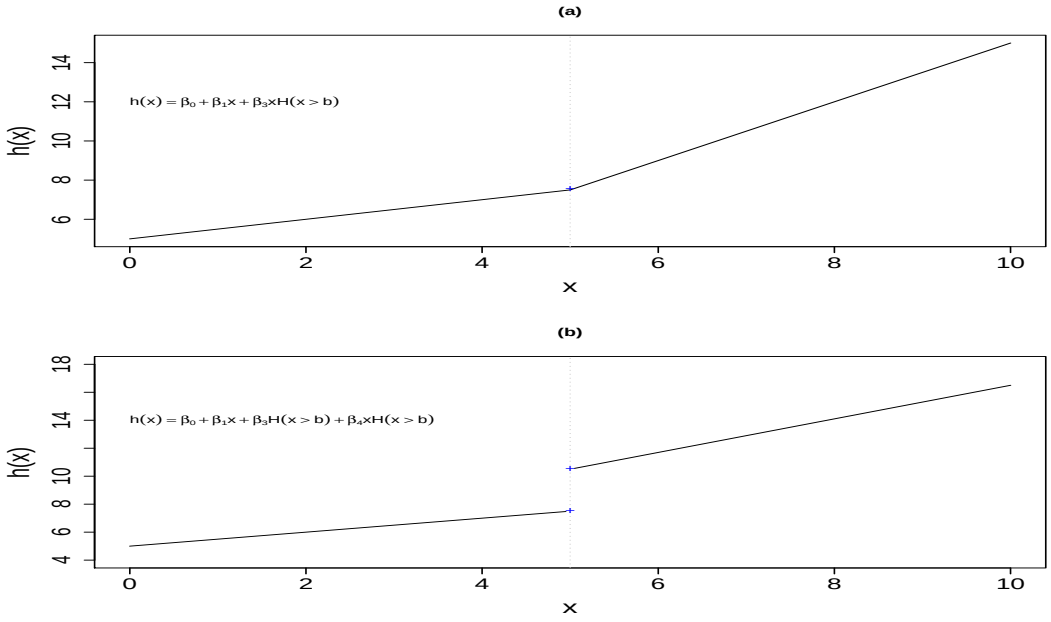


Figure 8.5: Piecewise linear, (a) continuous and (b) discontinuous lines.

discontinuous split line, the function has the form

$$h(x) = \beta_{00} + \beta_{01}x + [\beta_{10} + \beta_{11}(x - b)]\, H(x > b) \tag{8.3}$$

with β_{00}, β_{10}, β_{01}, β_{11}, and b being the parameters which need to be estimated. In real data exhibiting piecewise linear behaviour the breakpoint parameter b is usually the parameter of interest. Figure 8.5(b) shows a discontinuous split line curve with parameters $\beta_{00} = 5$, $\beta_{01} = 0.5$, $\beta_{10} = 3$, $\beta_{11} = 0.7$ and $b = 5$.

A quadratic piecewise polynomial with one breakpoint will have the form

$$h(x) = \beta_{00} + \beta_{01}x + \beta_{02}x^2 + \left[\beta_{10} + \beta_{11}(x - b) + \beta_{12}(x - b)^2\right] H(x > b). \tag{8.4}$$

The function is discontinuous at the breakpoint its first and second derivatives discontinuous, [see Figure 8.6(a)]. By dropping the term β_{10} from the equation the function becomes continuous but still has a discontinuous first and second derivatives at the breakpoint, [see Figure 8.6(b)]. The function becomes continuous and with continuous first derivative when the term $\beta_{11}(x-b)$ is dropped, [see Figure 8.6(c)]. To create Figure 8.6 the following values for the parameters were used: $\beta_{00} = 5$, $\beta_{01} = -0.1$, $\beta_{02} = 0.1$, $\beta_{10} = 2$, $\beta_{11} = 1$,$\beta_{12} = -0.4$ and $b = 5$.

More general piecewise polynomials are defined as

$$h(x) = \sum_{j=0}^{D} \beta_{0j}x^j + \sum_{k=1}^{K}\sum_{j=0}^{D} \beta_{kj}(x - b_k)^j H(x > b_k) \tag{8.5}$$

where D is the degree of the polynomial in x and K is the number of break points $\mathbf{b}$. The presence or absence of the term $\beta_{kj}(x - b_k)^j$ in the above equation allows a discontinuity or

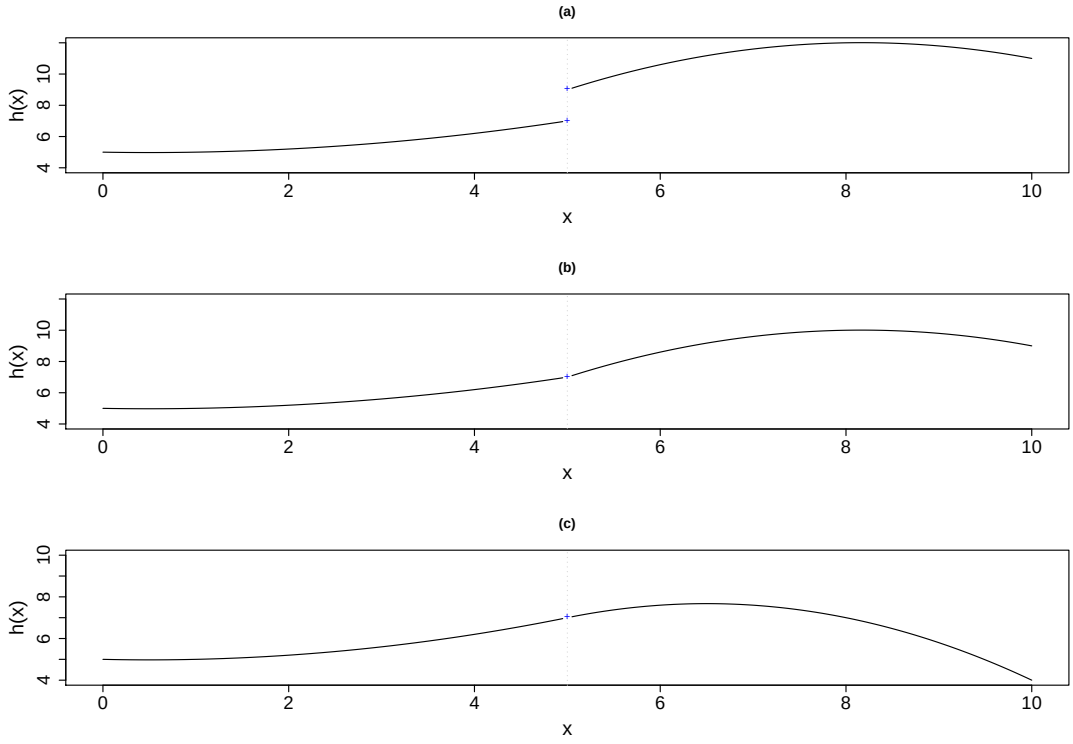


Figure 8.6: Piecewise quadratic, (a) discontinuous and discontinuous first derivative, (b) continuous with discontinuous first derivative and (c) continuous with continuous first derivative.

continuity respectively at the break point b_k in the jth derivative of the function. If continuity is required in the jth derivative of the function at a particular breakpoint b_k it would probably be required also in all the lower-order derivatives at b_k. This would be achieved by removing all the terms $\beta_{km}(x - b_k)^m$, for $m = 0, 1, \ldots k$, from the equation. The name *spline* is usually applied to piecewise polynomials with all the lower derivatives than D continuous at b_k. For example

$$h(x) = \sum_{j=0}^{D} \beta_{0j} x^j + \sum_{k=1}^{K} \beta_k (x - b_k)^D H(x > b_k) \tag{8.6}$$

is a spline function of degree D. For D=3 we have the *cubic splines*

$$h(x) = \beta_{00} + \beta_{01} x + \beta_{02} x^2 + \beta_{03} x^3 + \sum_{k=1}^{K} \beta_k (x - b_k)^3 H(x > b_k) \tag{8.7}$$

Cubic splines are, because of their continuous first and second derivatives at the break points, very smooth curves and therefore ideal for smoothing techniques.

In order to fit a spline curve as in equation (8.7) within a regression models you would need K non-linear b_k break point parameters and $K + 4$ linear β parameters to completely specified the equation. A design $\mathbf{X}$ matrix basis based on equation (8.7) is called a *truncated piecewise polynomial* basis. Figure 8.7 shows a truncated piecewise polynomial basis for degrees of polynomials equal to 0 constant, 1 linear, 2 quadratic and 3 cubic. The x-variable here is in the range from zero to one and there are five knots at $(0.2, 0.3, 0.5, 0.7, 0.8)$.

For degree = 0, in Figure 8.7(a), the basis functions comprise six dummy variables corresponding to the intervals $(0 < x \leq 0.2)$, $(0, 2 < x \leq 0.3)$, $(0.3 < x \leq 0.5)$ $(0.5 < x \leq 0.7)$ $(0.7 < x \leq 0.8)$, $(0.8 < x \leq 1)$, having ones if the value of x belongs to the interval and zero otherwise. For degree = 1 the basis functions, in Figure 8.7(b), comprise the constant plus six extra linear functions. The first linear function is defined on the whole range of x while the rest five only on a limited range, e.g. the second on the range $(0.2 < x < 1)$. Figure 8.7(c) shows the basis functions for degree= 2. Here we have, the constant, the linear plus 6 quadratic functions. The constant linear and first quadratic functions are define on the whole range of x while the other 5 quadratics functions are defined on a limited range of x depending on the break points (or knots). The same pattern appears in Figure 8.7(c) for degree= 3 where the basis functions comprise of the constant, linear, quadratic and cubic functions defined on the whole range of x while the other five cubic function are defined only on a limited range depending on the knots. While the truncated basis of a spline function is intuitively simple, it suffers with the same problem as a polynomial basis in that it is not numerically stable. The B-splines introduced in the next section is numerically superior.

8.6 B-Splines basis

B-splines are to the truncated piecewise polynomials what orthogonal polynomials are to polynomials, that is, an B-spline basis provides a superior numerical basis to equation (8.7). The basic functions in B-spline are defined only locally in the sense that they are non-zero only on the domain spanned by $2 + D$ knots, where D is the degree of the piecewise polynomial, see de Boor [1978] for further details. The term "B-spline" is short for basis spline. The important

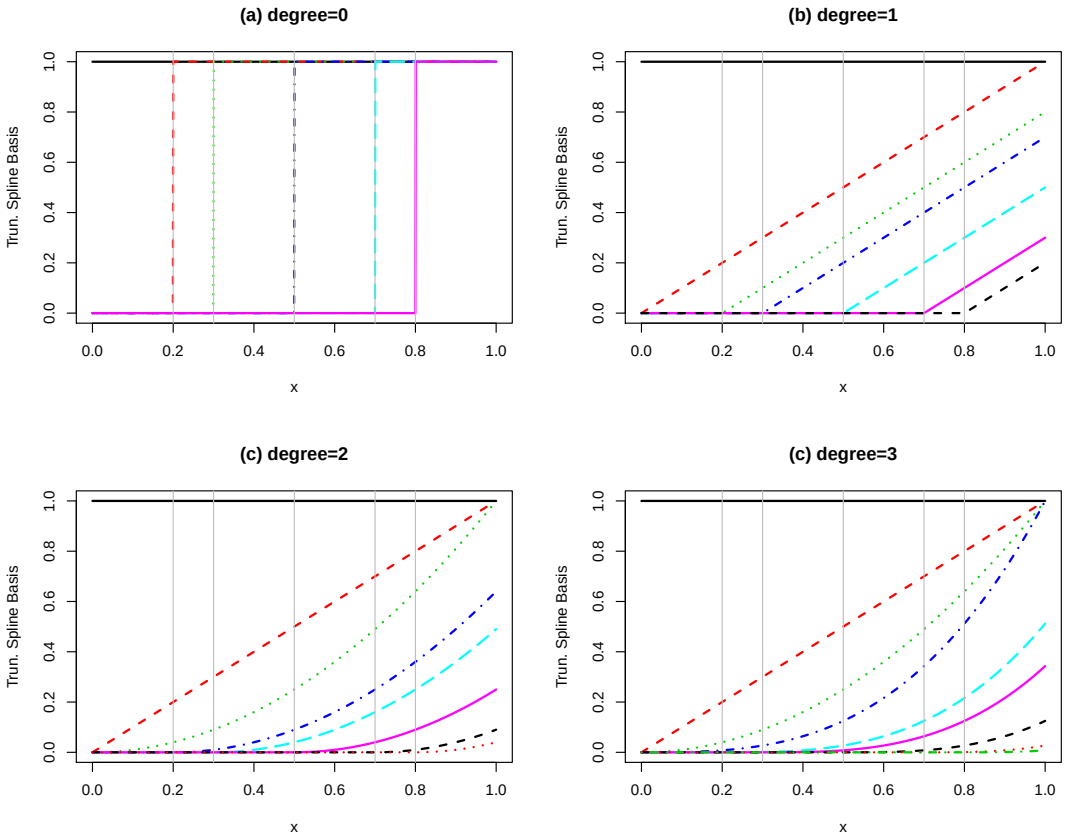


Figure 8.7: Showing truncated piecewise polynomials basis functions for different degrees a) constant, b) linear, c) quadratic and d) cubic, The x variable is defined from zero to one having break points at $(0.2, 0.4, 0.5, 0.6, 0.8)$.

thing here is that any function given by equation (8.7) of a given degree and for a given x range can be uniquely represented as a linear combination of B-splines of the same degree within the same range. There are several properties of B-splines worth noting:

- The B-splines are defined by local functions which have their domain within $2 + D$ knots of the x range. For example for cubic splines with $D = 3$ each base function is defined within 5 knots.
- Depending on the degree of the piecewise polynomial the B-splines could be
  - local constants ($D = 0$) function of x,
  - local (two piece) linear ($D = 1$) function of x,
  - local (three piece) quadratic ($D = 2$) function of x,
  - local (four piece) cubic ($D = 3$) function of x or
  - any higher level (four $D + 1$) polynomial $D => 4$
    Figure 8.8 show an example for $D = 0, 1, 2, 3$. Note that the basis functions of a cubic spline are very similar in shape to the normal distribution.
- The knots do not have to be in equal distance, so general patterns of knots are possible.
- The number of knots determines the size of B-spline basis which make up the piecewise polynomial function.
- B-splines are columns of basis matrix $\mathbf{B}$. This matrix can be used in a regression framework as the design matrix. The fitted coefficients in such regression $\hat{\mathbf{y}} = \mathbf{B}\hat{\boldsymbol{\beta}}$ produce a flexible non-linear relationship between y and x. Figure 8.9 shows a regression fit using the `aids` data. Figure 8.9(a) shows the B-spline basis functions with $D = 3$ for fitting x (time) generated with 8 equal space knots. Figure 8.9(b) shows the fitted spline (solid line) and the basis functions this time weighted by their fitted coefficients.

Models in which the break point parameters (or knots) b_k are determine in advance, so only the linear parameters β have to be estimated, are called *regression spline* models. If the position of the knots (or the break points) is chosen uniformly over the range of the x-variable then the regression spline are called *Cardinal splines*. Another method of regression splines is the one which uses positions determined from the quantile values the x-variable. The number of knots in both procedures effects the degrees of freedom for the fitted model and therefore the complexity of the model. Parameters which determine the complexity of the model are usually referred to as *smoothing parameters*. The degrees of freedom for the fitted model in this case are $1 + D + K$, One for the intercept, D for the degree of the polynomial in x and K for the number of knots. This design matrix $\mathbf{B}$ has $1 + D + K$ independent columns. To create a B-splines basis, the function `bs()` or `ns()` from package **splines** can be used. The first one generates the B-spline basis matrix for a polynomial spline of any degree and the second generates the B-spline basis matrix for a *natural* cubic spline. Natural cubic splines are cubic splines having the extra condition that the behaviour of the function outside the range of x is linear. An example of using `bs()` is given in Section 8.8.3.

Models in which the break points have to be estimated are called *free knot* models and they are examined in next section.

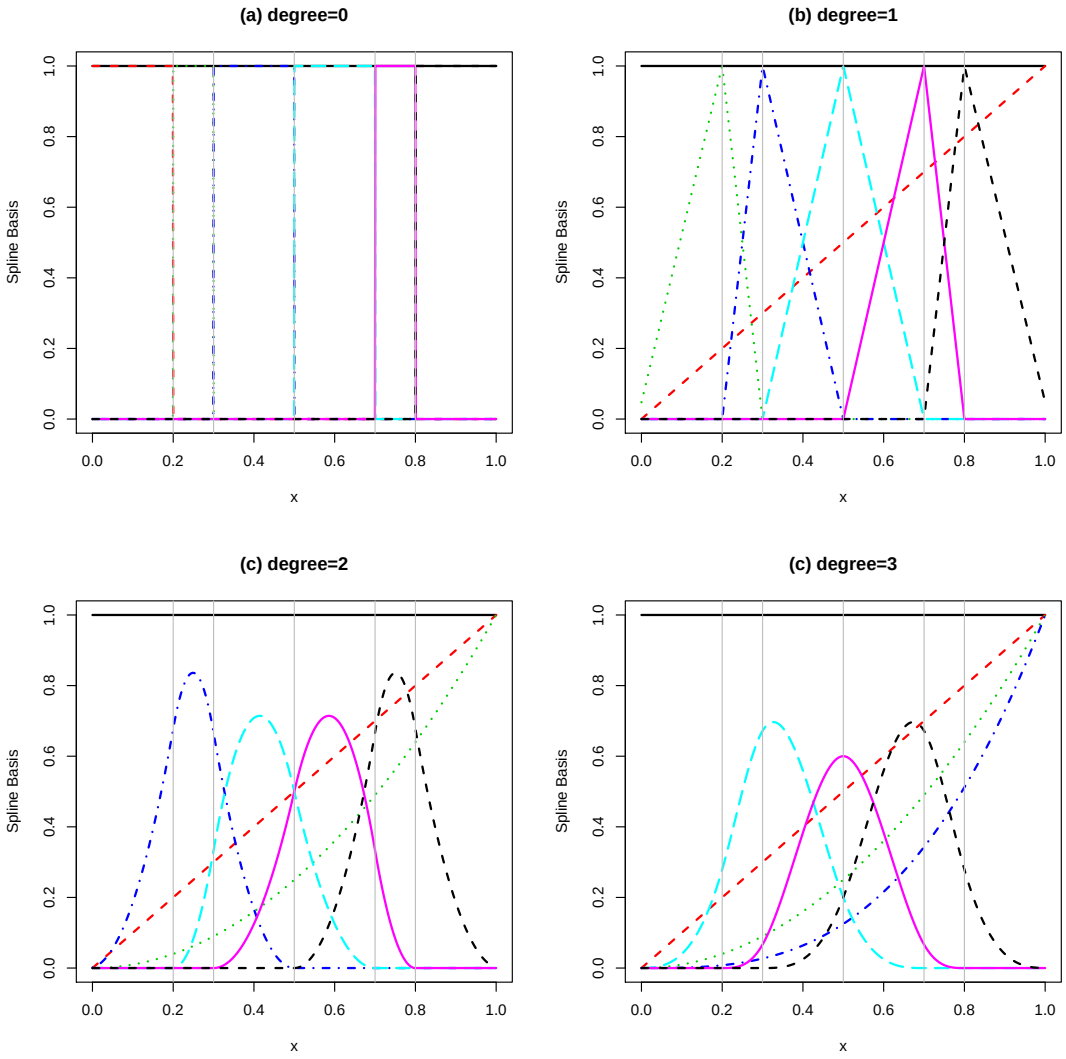


Figure 8.8: Showing B-spline basis for different degrees a) constant, b) linear, c) quadratic and d) cubic, The x variable is defined from zero to one having unequal spaced knots (break points) at $(0.2, 0.4, 0.5, 0.6, 0.8)$.

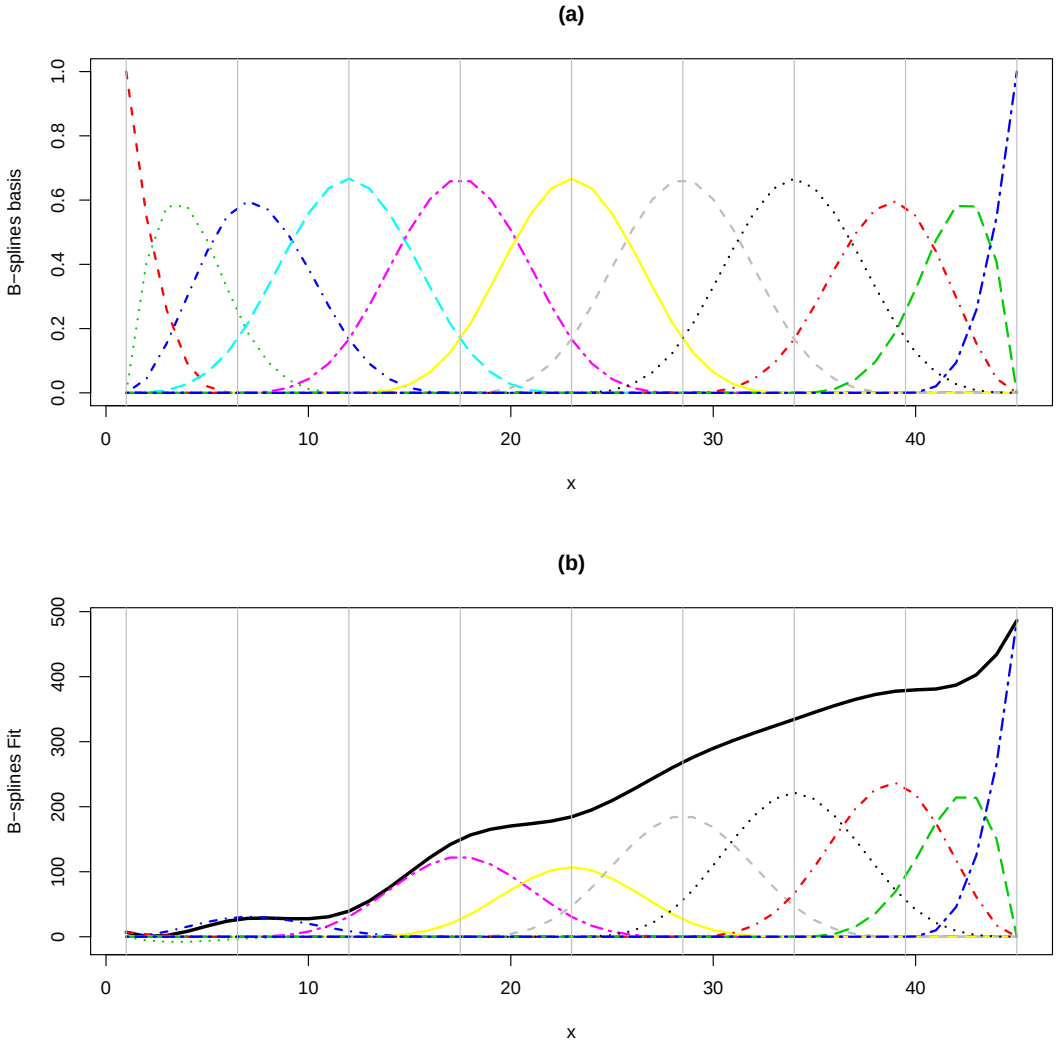


Figure 8.9: Showing B-splines fit of y (the number of aids cases) against x (time) for the `aids` data using 8 equal space knots. a) Showing the B-splines basis for x, and b) showing the fitted values for y in black plus the B-splines basis functions weighted by their coefficients $\hat{\boldsymbol{\beta}}$.

8.7 Free knots break point models

Free knots (or break point) models are piecewise polynomial models where the position and number of knots have to estimated from the data. The models are useful if it is believed that there is a structural change or changes in the relationship between the y and x and that the time(s) or point(s) where the relationship changes are unknown. Estimation of the break points is a highly non-linear problem. The likelihood function of the knot parameters is notorious for its multiple maxima. In this section we concentrate on cases where there are relatively few knots. For example Figure **??**(a) is a typical example where at some point the linear relationship between y and x is changing changes.

The `gamlss.add` packages provide few functions for break point modelling:

- `fitFixedKnots()` : for fitting a univariate regression model using piecewise polynomials with known knots
- `fitFreeKnots()`: for fitting a univariate regression model using piecewise polynomials with unknown knots
- `fk()`: for fitting a regression additive terms using piecewise polynomials with unknown (or known) knots

The argument for the functions `fitFixedKnots()` and `fitFreeKnots()` are:

`x` the x variable (explanatory)

`y` the response variable

`weights` the prior weights

`knots` the position of the interior knots for `fitFixedKnots()` or starting values for `fitFreeKnots()`

`data` the data frame

`degree` the degree of the piecewise polynomials

`base` The basis functions for the piecewise polynomials, `"trun"`, for truncated (default), and `"Bbase"` for B-spline basis piecewise polynomials

`trace` controlling the trace of of `optim()`, only used for the function `fitFreeKnots()`

`...` for extra arguments

Those two functions return a S3 class object `"FreeBreakPointsReg"` and `"FixBreakPointsReg"` respectively. This objects have methods `print()`, `fitted()`, `residuals()`, `coef()`, `knots()` and `predict()`. The `"FixBreakPointsReg"` objects also have `vcov` and `summary()`. The function `fk()` provides an interface so those two functions can be utilised within gamlss(). The main arguments of the function `fk()` are:

`x` the x variable

`start` starting values for the breakpoints. The number of break points is also determined by the length of start.

while other arguments for `"FreeBreakPointsReg"` or `"FixBreakPointsReg"` can be pass through `control`. An example of using the `fk()` function is given in Section 8.8.4.

8.8 Example: the CD4 data

Data summary: the CD4 data

R data file: CD4 in package **MASS** of dimensions 609×2

variables

cd4 : CD4 counts from uninfected children born to HIV-1 mothers.

age : The age of child in in years

purpose: to demonstrate the use of linear parametric terms

This section gives an example on how some of the techniques described in the previous section can be used. The data are given by Wade and Ades (1994) and they refer to cd4 counts from uninfected children born to HIV-1 mothers and the age in years of the child. Here we input and plot the data in Figure **??**. This is a simple regression example with only one explanatory variable, the age, which is a continuous variable. The response while, strictly speaking is a count, is sufficiently large for us to treat it at this stage as a continuous response variable.

Figure 8.10

```
data("CD4")
plot(cd4 ~ age, data = CD4)
```

R code on page 198

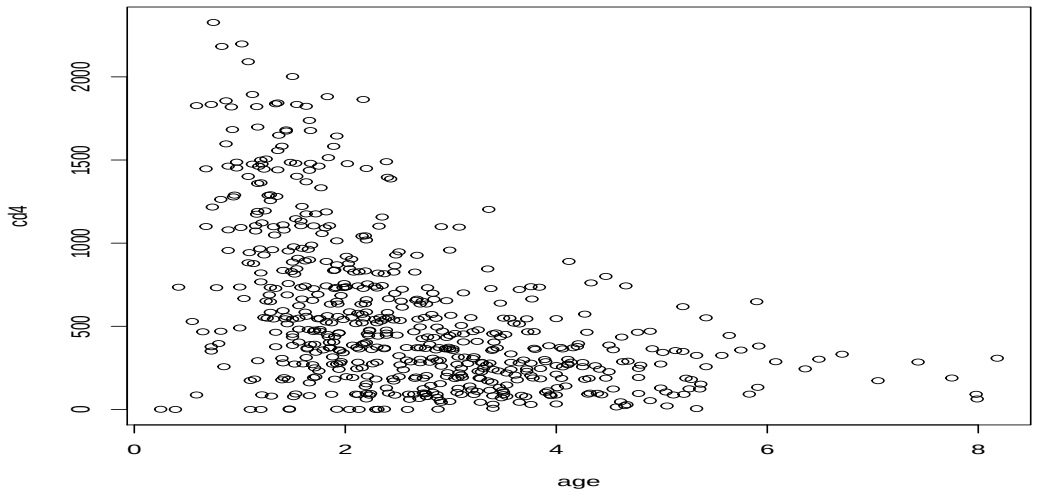


Figure 8.10: The cd4 data.

There are several striking features in this specific set of data in Figure **??**.

1. The first has to do with the relationship between the mean of cd4 and age. It is hard to see from the plot whether this relationship is linear or not.
2. The second has to do with the heterogeneity of variance in the response variable cd4. It

appears that the variation in cd4 is decreasing with age.

3. The final problem has to do with the distribution of cd4 given the age. Is this distribution normal? It is hard to tell from the figure but probably we will need a more flexible distribution.

Traditionally, problems of this kind were dealt with by a transformation in the response variable or a transformation in both in the response and the explanatory variable(s). One could hope that this would possibly correct for some or all the above problems simultaneously. Figure ?? (produced with the following code) shows plots where several transformations for cd4 and age were tried. It is hard to see how we can improve the situation by transformations.

Figure 8.11

```
op <- par(mfrow = c(3, 4), mar = par("mar") + c(0, 1, 0, 0),
     pch = "+", cex = 0.45, cex.lab = 1.8, cex.axis = 1.6)
page <- c("age^-0.5", "log(age)", "age^.5", "age")
pcd4 <- c("cd4^-0.5", "log(cd4+1)", "cd4^.5")
for (i in 1:3) {
     yy <- with(CD4, eval(parse(text = pcd4[i])))
     for (j in 1:4) {
         xx <- with(CD4, eval(parse(text = page[j])))
         plot(yy ~ xx, xlab = page[j], ylab = pcd4[i])
     }
 }
par(op)
```

Within the GAMLSS framework we can deal with these problems one at the time. First we start with the relationship between the mean of cd4 and age.

8.8.1 Orthogonal polynomials

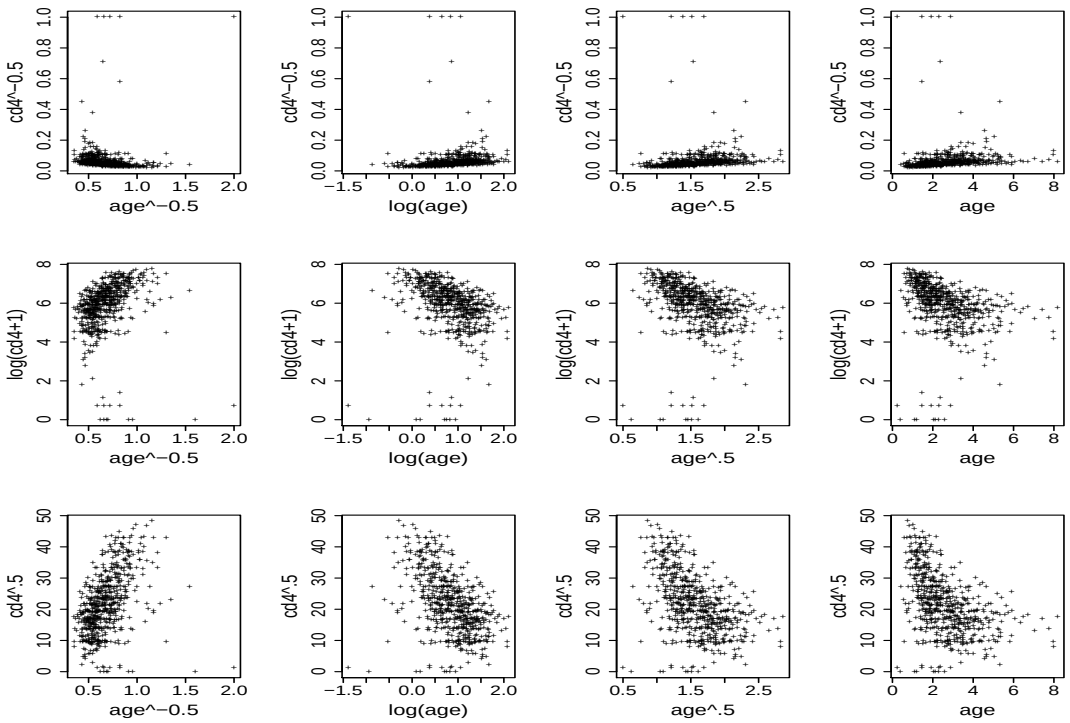
We will fit orthogonal polynomials of different orders to the data and choose the best using a GAIC criterion. For now we fit a constant variance and a default normal distribution.

```
m1 <- gamlss(cd4 ~ age, sigma.fo = ~1, data = CD4, trace = FALSE)
m2 <- gamlss(cd4 ~ poly(age, 2), sigma.fo = ~1, data = CD4, trace = FALSE)
m3 <- gamlss(cd4 ~ poly(age, 3), sigma.fo = ~1, data = CD4, trace = FALSE)
m4 <- gamlss(cd4 ~ poly(age, 4), sigma.fo = ~1, data = CD4, trace = FALSE)
m5 <- gamlss(cd4 ~ poly(age, 5), sigma.fo = ~1, data = CD4, trace = FALSE)
m6 <- gamlss(cd4 ~ poly(age, 6), sigma.fo = ~1, data = CD4, trace = FALSE)
m7 <- gamlss(cd4 ~ poly(age, 7), sigma.fo = ~1, data = CD4, trace = FALSE)
m8 <- gamlss(cd4 ~ poly(age, 8), sigma.fo = ~1, data = CD4, trace = FALSE)
```

First we compare the models using the Akaike Information criterion (AIC) which has penalty $k = 2$ for each parameter in the model, (the default value in the function GAIC()):

```
GAIC(m1, m2, m3, m4, m5, m7, m8)
```

```
##    df      AIC
## m7  9 8963.263
## m8 10 8963.874
## m5  7 8977.383
```



R code on page 199

Figure 8.11: The CD4 data with various transformations for `cd4` and age

```
## m4  6 8988.105
## m3  5 8993.351
## m2  4 8995.636
## m1  3 9044.145
```

Next we compare the models using Schwartz Bayesian Criterion (SBC) which uses penalty $k = \log(n)$:

```
GAIC(m1, m2, m3, m4, m5, m7, m8, k = log(length(CD4$age)))
```

```
##    df      AIC
## m7  9 9002.969
## m8 10 9007.992
## m5  7 9008.266
## m2  4 9013.284
## m4  6 9014.576
## m3  5 9015.410
## m1  3 9057.380
```

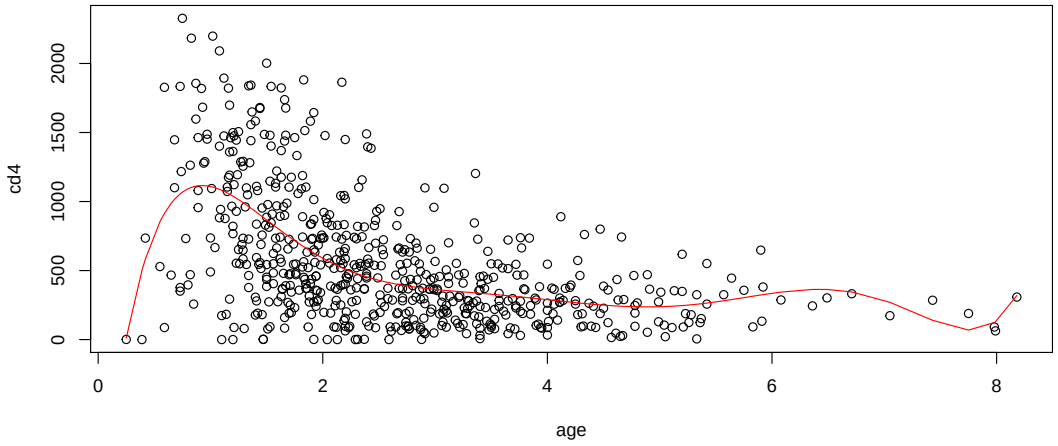


Figure 8.12: The CD4 data and the fitted values using polynomial of degree 7 in age

Remarkably with both AIC and SBC select model m7, with a polynomial of degree 7, as the best model. Unfortunately the fitted values for the mean of `cd4` shown together with the data in Figure 8.12 look rather unconvincing. The line is too wobbly at the ends of the range of `age`, trying to be very close to the data. This is a typical behaviour of polynomial fitting.

8.8.2 Fractional polynomials

Now we will try alternatives methods, two parametric, using *fractional polynomials*. Fractional polynomials were introduced by Royston and Altmam (1994). The function `fp()` which we are going to use to fit them works in `gamlss()` as an additive smoother term. It can be used to

fit the best (fractional) polynomial within a specific set of possible power values. Its argument `npoly` determines whether one, two or three terms in the fractional polynomial will be used in the fitting. Here we fit fractional polynomials with one, two and three terms respectively and we choose the best using GAIC:

```
m1f <- gamlss(cd4 ~ fp(age, 1), sigma.fo = ~1, data = CD4, trace = FALSE)
m2f <- gamlss(cd4 ~ fp(age, 2), sigma.fo = ~1, data = CD4, trace = FALSE)
m3f <- gamlss(cd4 ~ fp(age, 3), sigma.fo = ~1, data = CD4, trace = FALSE)
GAIC(m1f, m2f, m3f)
```

```
##     df      AIC
## m3f  8 8966.375
## m2f  6 8978.469
## m1f  4 9015.321
```

```
GAIC(m1f, m2f, m3f, k = log(length(CD4$age)))
```

```
##     df      AIC
## m3f  8 9001.669
## m2f  6 9004.940
## m1f  4 9032.968
```

```
# to get the fitted GAMLSS model
m3f
```

```
##
## Family:  c("NO", "Normal")
## Fitting method: RS()
##
## Call:
## gamlss(formula = cd4 ~ fp(age, 3), sigma.formula = ~1, data = CD4,
##     trace = FALSE)
##
## Mu Coefficients:
## (Intercept)   fp(age, 3)
##       557.5           NA
## Sigma Coefficients:
## (Intercept)
##       5.929
##
##  Degrees of Freedom for the fit: 8 Residual Deg. of Freedom   601
## Global Deviance:     8950.37
##             AIC:     8966.37
##             SBC:     9001.67
```

```
# to get the fitted fractional polynomial (note that it is a lm class object)
getSmo(m3f)
```

```
##
## Call:
## lm(formula = y ~ x.fp, weights = w)
##
```

```
## Coefficients:
## (Intercept)        x.fp1        x.fp2        x.fp3
##      -599.3       1116.8       1776.2        698.6

# to get the power parameters
getSmo(m3f)$power

## [1] -2 -2 -2
```

```
plot(cd4 ~ age, data = CD4)
lines(CD4$age[order(CD4$age)], fitted(m1f)[order(CD4$age)],
      lty=1, col = "blue")
lines(CD4$age[order(CD4$age)], fitted(m2f)[order(CD4$age)],
      lty=2, col = "green")
lines(CD4$age[order(CD4$age)], fitted(m3f)[order(CD4$age)],
      lty=3, col = "red")
```

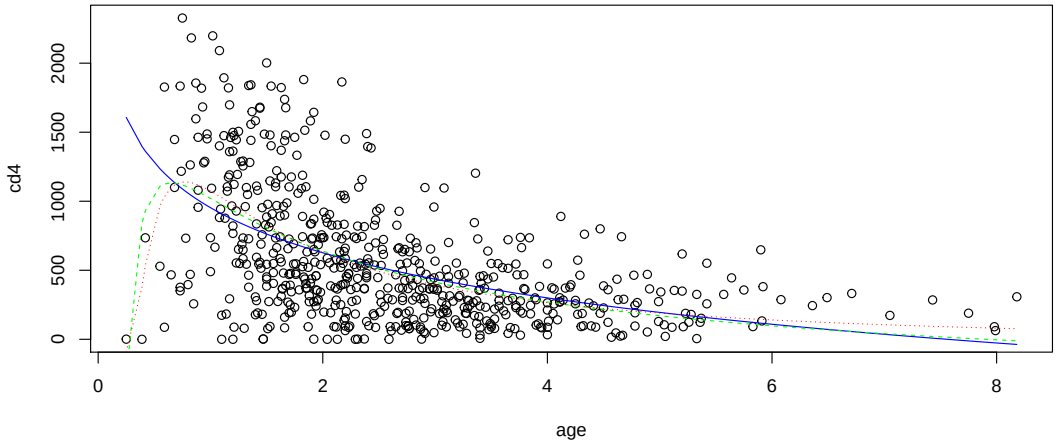


Figure 8.13: The CD4 data and the fitted values using fractional polynomial of degree 1 (solid), 2 (dashed), 3 (dotted) in age

Both AIC and BSC favour the model `m3f` with a fractional polynomial with three terms. Note that by printing `m3f` the model for μ gives a value of 557.5 for the "Intercept" and NULL for the coefficient for `fp(age, 3)`. This is because within the backfitting the constant is fitted first and then the fractional polynomial is fitted to the partial residuals of the constant model. As a consequence the constant is fitted twice. The coefficients and the power transformations of the fractional polynomials can be obtained using the `mu.coefSmo` component of the `gamlss` fitted object. For the `CD4` data all powers happens to be -2 indicating that the following terms are fitted in the model, age^{-2}, $age^{-2}\log(age)$ and $age^{-2}\left[\log(age)\right]^2$. Hence the fitted model `m3f` is given by $cd4 \sim NO(\hat{\mu}, \hat{\sigma})$, where $\hat{\mu} = 557.5 - 599.3 + 1116.8\, age^{-2} + 1776.2\, age^{-2}\log(age) + 698.6\, age^{-2}\left[\log(age)^2\right]$ and $\hat{\sigma} = \exp(5.929) = 375.8$. Figure 8.13 shows the best fitted models using one, two or three fractional polynomial terms. The situation remains unconvincing. None

of the models seem to fit particular well.

8.8.3 Piecewise polynomials

Next we will fit piecewise polynomials using the `R` function `bs`. We try different degrees of freedom (effectively different number of knots) and we choose the best model using AIC and SBC:

```
m2b <- gamlss(cd4 ~ bs(age), data = CD4, trace = FALSE)
m3b <- gamlss(cd4 ~ bs(age, df = 3), data = CD4, trace = FALSE)
m4b <- gamlss(cd4 ~ bs(age, df = 4), data = CD4, trace = FALSE)
m5b <- gamlss(cd4 ~ bs(age, df = 5), data = CD4, trace = FALSE)
m6b <- gamlss(cd4 ~ bs(age, df = 6), data = CD4, trace = FALSE)
m7b <- gamlss(cd4 ~ bs(age, df = 7), data = CD4, trace = FALSE)
m8b <- gamlss(cd4 ~ bs(age, df = 8), data = CD4, trace = FALSE)
GAIC(m2b, m3b, m4b, m5b, m6b, m7b, m8b)
```

```
##     df      AIC
## m7b  9 8959.519
## m6b  8 8960.353
## m8b 10 8961.073
## m5b  7 8964.022
## m4b  6 8977.475
## m2b  5 8993.351
## m3b  5 8993.351
```

```
GAIC(m2b, m3b, m4b, m5b, m6b, m7b, m8b, k = log(length(CD4$age)))
```

```
##     df      AIC
## m5b  7 8994.904
## m6b  8 8995.648
## m7b  9 8999.225
## m4b  6 9003.946
## m8b 10 9005.191
## m2b  5 9015.410
## m3b  5 9015.410
```

Note that model `m2b` uses the default $df = 3$. The best model with AIC uses 7 degrees of freedom while SBC uses 5. Figure 8.14 shows the fitted models using 5 and 7 degrees of freedom for the piecewise polynomial in age.

```
plot(cd4 ~ age, data = CD4)
lines(CD4$age[order(CD4$age)], fitted(m7b)[order(CD4$age)], col = "blue",
      lty=1, lwd=2)
lines(CD4$age[order(CD4$age)], fitted(m5b )[order(CD4$age)], col = "red",
      lty=2, lwd=2)
```

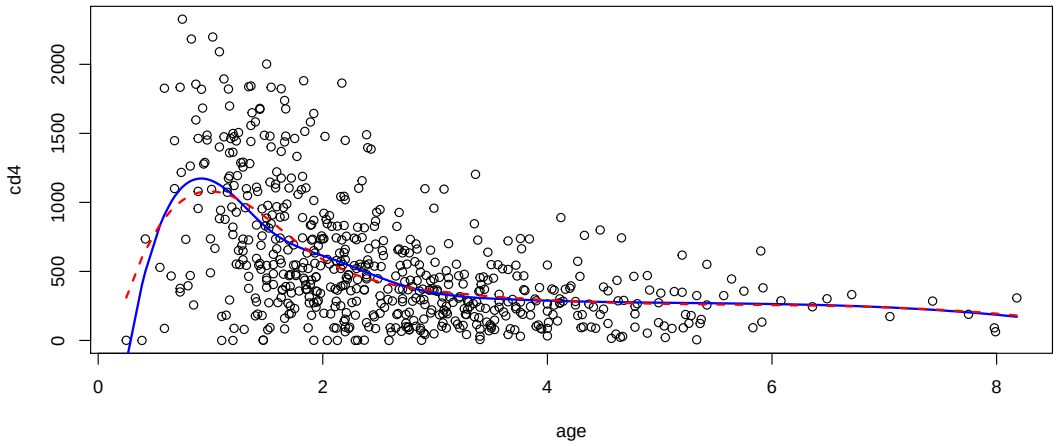


Figure 8.14: The CD4 data and the fitted values using piecewise polynomial with degrees of freedom 5 (dashed line) and 7 (solid line) for age

8.8.4 Free knots

Here we are trying to model the relationship between `cd4` and `age` using simple piecewise polynomials. We fit four different models using linear (`degree=1`) and quadratic functions (`degree=2`) and using one and two break points respectively. Note that starting values have to be specified for the break points. The number of break points to fit is taken from the number of starting values.

```
library(gamlss.add)
 f1<-gamlss(cd4~fk(age, degree=1, start=2), data=CD4, trace = FALSE)
 f2<-gamlss(cd4~fk(age, degree=1, start=c(2,5)), data=CD4, trace = FALSE)
 f3<-gamlss(cd4~fk(age, degree=2, start=2), data=CD4, trace = FALSE)
 f4<-gamlss(cd4~fk(age, degree=2, start=c(2,5)), data=CD4, trace = FALSE)
GAIC(f1, f2, f3, f4)
```

```
##    df      AIC
## f1  5 8984.558
## f3  5 8984.558
## f2  7 8988.357
## f4  7 8988.357
```

```
GAIC(f1, f2, f3, f4, k = log(length(CD4$age)))
```

```
##    df      AIC
## f1  5 9006.617
## f3  5 9006.617
## f2  7 9019.239
## f4  7 9019.239
```

From the GAIC it can be seen that there is no support for the quadratic models `f3` and `f4` and that the data support only one break point parameter. To get the two different slopes and the break point parameters use the `getSmo()` function as it illustrated below.

```
f1
```

```
##
## Family:  c("NO", "Normal")
## Fitting method: RS()
##
## Call:
## gamlss(formula = cd4 ~ fk(age, degree = 1, start = 2), data = CD4,
##     trace = FALSE)
##
## Mu Coefficients:
##                     (Intercept)  fk(age, degree = 1, start = 2)
##                           557.5                              NA
## Sigma Coefficients:
## (Intercept)
##       5.949
##
##  Degrees of Freedom for the fit: 5 Residual Deg. of Freedom   604
## Global Deviance:     8974.56
##             AIC:     8984.56
##             SBC:     9006.62
```

```
getSmo(f1)
```

```
##
## Call:
## fitFreeKnots(y = y, x = xvar, weights = w, degree = degree, knots = lambda,
##     fixed = control$fixed, base = control$base)
##
## Coefficients:
## (Intercept)            x       XatBP1
##       809.1       -361.5        334.9
## Estimated Knots:
##  BP1
## 2.87
```

The break point can also be found just using

```
knots(getSmo(f1))
```

```
##      BP1
## 2.869966
```

The fitted linear plus linear model for μ is given

$$\begin{aligned}\mu = \eta_1 &= (557.5 + 809.1) - 361.5\,\texttt{age} + 334.9\,\texttt{age}\,\texttt{if}(\texttt{age} > 2.869) \\ &= 1366.6 - 361.5\,\texttt{age}\,\texttt{if}(\texttt{age} \leq 2.869) - 26.6\,\texttt{age}\,\texttt{if}(\texttt{age} > 2.869)\end{aligned}$$

The plot of the fitted model is given in Figure 8.15.

```
plot(cd4 ~ age, data = CD4)
lines(CD4$age[order(CD4$age)], fitted(f1)[order(CD4$age)], col = "blue", lty=1,
      lwd=2)
```

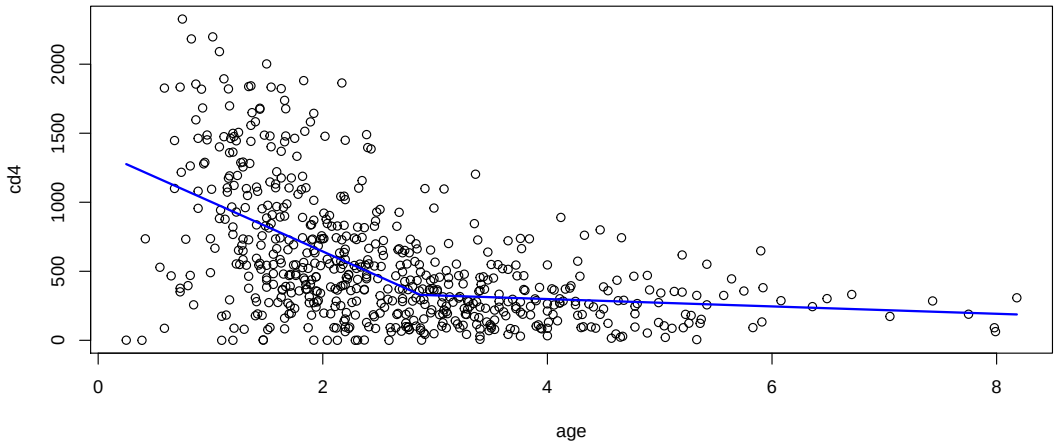


Figure 8.15: The CD4 data and the fitted values using piecewise linear fit with the knot estimated from the data

Chapter 9

Additive Smoothing Terms

This chapter provides an introduction to smoothing techniques and how to use those techniques within a GAMLSS model. In particular:

- univariate penalised smoothing techniques are introduced using local fits,
- the penalised approach to smoothing is explained,
- the GAMLSS smoothing additive terms are described.

9.1 Introduction

This chapter is dedicated to smoothing techniques and how they can be applied and used within the GAMLSS framework. A univariate smoother, $f(x)$ is where only one explanatory variable x is used. A multivariate smoother, say $f(x_1, x_2)$, is defined where two or more explanatory variable are involved in the fitting. Both univariate and multivariate smoothers can be used as additive terms with a GAMLSS model formula. We can think of the univariate smoothers as the *main* (non-linear) effects of the explanatory variables on a distribution parameter while the multivariate smoothers as their non-linear *interaction* effects.

We will classify all smoothers used within GAMLSS into two main categories:

the penalised smoothers: which use quadratic penalties on the fitted smooth model parameters to control the amount of smoothing and

all others smoothers: which use different ideas (i.e. locality) or non-quadratic penalties to achieve the resulting smooth functions.

The distinction is illustrated in the diagram of Figure **??**, where also the difference between univariate and multivariate smoothers is highlighted.

The structure of this section is as follows. Section 9.2 is an introduction to smoothing techniques in general. Section 9.3 describes local regression smoothers and serves as an introduction to basic ideas in smoothing, like smoothing parameters and locality of the estimates. Sections 9.4

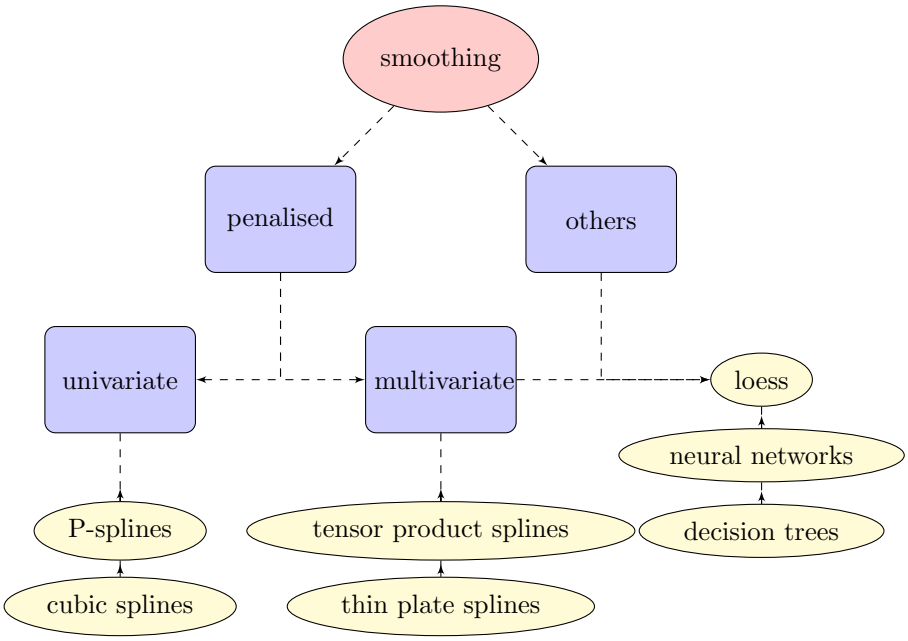


Figure 9.1: Diagram showing the different additive smoothing terms in GAMLSS

and 9.5 explain how the univariate and multivariate penalised smoothers can be used within GAMLSS, respectively. The "other" smoothers are explained in section 9.6.

9.2 What is a scatterplot smoother

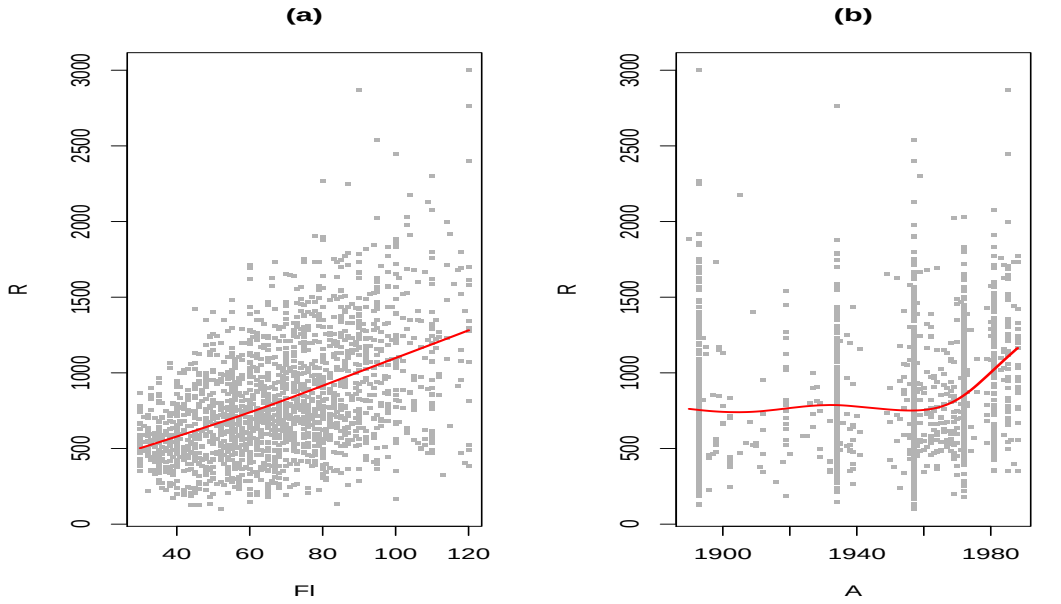
Suppose we have n measurements of a response variable $\mathbf{y} = (y_1, y_2, \ldots, y_n)^\top$ and a single explanatory variable $\mathbf{x} = (x_1, x_2, \ldots, x_n)^\top$ and we want to study their relationship. The first thing we should do is to plot the variables y (vertically) against x (horizontally). A curve fitted through the data show the kind of relationship existing between the two variables.

For demonstration purposes we return to the he Munich 1990's rent data fist introduce in Chapter **??**. The Figure 9.2(a) is a plot of the `rent` against floor space, `Fl`, and 9.2(b) is a plot of the `rent` against the age (i.e. year of construction) of the building, `A`. If we ignore the fitted line smoothers shown in the plots for the moment, the left plot shows a clear positive relationship between rent and floor space but between rent and age the relationship is not clear cut.

Figure 9.2

```
data(rent)
m1 <- gamlss(R~pb(Fl), data=rent)

## GAMLSS-RS iteration 1: Global Deviance = 28460.85
## GAMLSS-RS iteration 2: Global Deviance = 28460.85

m2  <- gamlss(R~pb(A), data=rent)

## GAMLSS-RS iteration 1: Global Deviance = 28831.01
```

```
## GAMLSS-RS iteration 2: Global Deviance = 28831.01

op<-par(mfrow=c(1,2))
plot(R~Fl, data=rent, pch = 15, cex = 0.5, col = gray(0.7), main="(a)")
lines(fitted(m1)[order(rent$Fl)]~rent$Fl[order(rent$Fl)], col="red", lwd=2)
plot(R~A, data=rent, pch = 15, cex = 0.5, col = gray(0.7), main="(b)")
lines(fitted(m2)[order(rent$A)]~rent$A[order(rent$A)], col="red", lwd=2)
par(op)
```



R code on page 210

Figure 9.2: The Munich 90's rent data set: a) rent prices against floor space b) rent places against age of the building with smooth curves fitted

Scatterplot Smoothers are statistical devices which could help us to fit curves in situations like this.

Definition: A *scatterplot smoother*, or for convenience, a *smoother* summarizes the trend of the response variable as a function of $\mathbf{x}$ by not assuming any parametric functional form for the dependence of $\mathbf{y}$ on $\mathbf{x}$.

It is this help in interpreting relationships which makes the smoothers important in statistics. For example, the fitted smoother in Figure 9.2(a) confirms our belief of a positive (almost linear) relationship between the rent and the floor space. From the smoother in Figure 9.2(b), we can conclude that for flats build between the 1900 to 1960 the rent values are relatively constant while there is a strong positive relationship between the rent and the age of the building after the 1960.

The following example shows that smoothing is also helpful in cases where the response variable

is not a continuous variable but binary. Consider the data in Figure 9.3, kindly provided by Prof Brian Francis of Lancaster University. Here we have a scatter plot of 10590 observations The response variable is whether a particular crime was reported ($y = 1$) or not ($y = 0$) in the media. The explanatory variable is the age of the victim of crime. The scatter plot in this case (ignoring for the moment the fitted smoothing curve in the middle) is uninformative due to the nature of the data. The smoothing curve in the figure is obtained using the additive function `pb()` and a binomial error for the response variable. The curve shows the fitted or estimated probability of reporting a crime in the media according to age. It shows that the estimated probability that a crime is reported is higher when the crime is committed on a young person, with a peak at around age ten. The estimated probability then declines until the victim reaches the age of twenty. From then on, the estimated probability, remains constant until the age of sixty after which the reporting probability rises steadily with age.

Figure 9.3

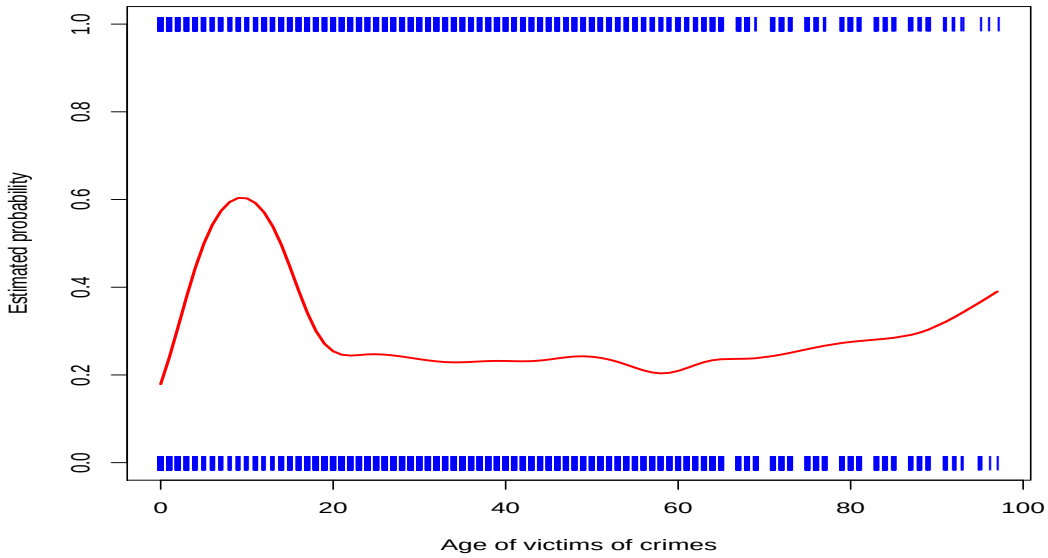
```
  data(VictimsOfCrime)
m1<- gamlss(reported~pb(age), data=VictimsOfCrime, family=BI)

## GAMLSS-RS iteration 1: Global Deviance = 11775.11
## GAMLSS-RS iteration 2: Global Deviance = 11775.13
## GAMLSS-RS iteration 3: Global Deviance = 11775.13

plot(reported~age, data=VictimsOfCrime, type="n",
     xlab="Age of victims of crimes", ylab="Estimated probability")
points(VictimsOfCrime$reported~jitter(VictimsOfCrime$age), pch="|", col="blue")
with(VictimsOfCrime, lines(fitted(m1)[order(age)]~age[order(age)],
                           col="red", lwd=2))
```

There was an explosion of statistical smoothing techniques at the late 80's and early 90's section and the reader is referred to books like Hastie and Tibshirani [1990], Green and Silverman [1994] Fahrmeir and Tutz [2001] Ruppert et al. [2003] Wang [2011] and Fahrmeir et al. [2013] for more details. The smoother originally was used to estimate of the conditional expected value of $\mathbf{y}$ given $\mathbf{x}$, $E(\mathbf{y}|\mathbf{x})$. This was extended over the years to any location parameter of the distribution of $\mathbf{y}$, e.g.. the median and more generally, smoothers are used for the estimation of quantiles or expectiles of the distribution for $\mathbf{y}$ given $\mathbf{x}$, Schnabel and Eilers [2013a,b] The important property of a smoother is that it does not assume a parametric functional form for the dependence between $\mathbf{y}$ and $\mathbf{x}$ but lets the data indicate the functional form. They are several ways to do that: for example to rely on local estimators or to put penalties on the behaviour of the parameters. The fact that they are called non-parametric is a bit misleading since all smoother do estimate parameters but also contain a dominant parameter which determines the amount of smoothing to the data. This parameter is called the *smoothing parameter*. How to chose the smoothing parameter is an essential issue in any smoothing technique.

A simple univariate smoother is a generalization of the simple linear regression model and can be written formally as a statistical model as:

$$E(Y_i) = a + f(x_i) \tag{9.1}$$

for $i = 1, \ldots, n$ where $f(.)$ is an arbitrary function which we assume to exist, α is a constant which most of the time we absorb into the function $f(.)$, say $\mu_i = f(x_i)$, and $f(x_i)$ is the trend that we would like to estimate. Typically, for a continuous response variables $Y_i \sim N(\mu_i, \sigma^2)$ for $i = 1, \ldots, n$ and that Y_i's are independent. The function $f(.)$ is arbitrarily defined but we assume, that it has some properties. For example a cubic spline smoother assumes that the function $f()$ has continuous first and second derivatives.



R code on page 212

Figure 9.3: Whether crime was reported in the media (1 =yes, 0 =no) against the age of the victim, together with smooth curve of the fitted probability crime was reported in the media.

An advantage of a smoother over a parametric fitting function is its local behaviour. Smoothers are effected by local observations more than by observations far way. Some of the basic ideas of smoothing can be introduced through local regression models and this is what next section is doing.

9.3 Local regression smoothers

The idea with the local smoother is that instead of using all available data to obtain a suitable estimate of the current value, only part of the data is used at a time. This part is determined by a *window*, which is an interval of the explanatory variable x. The window allows only observations that fall within it to count in the calculation of the current smoothed value. Except near the ends of the range of x, the window is a 'symmetric' two sided window where a neighbourhood of the target x is used with an equal number of observations on each side of the target value. Note that here we describe only a two sided window compared to one sided used extensively in time series analysis.

The bigger the window, the smoother the values of the estimates usually are (and the smaller the variance of the estimates). With smaller widows the estimates are more wiggly (but the estimates are less biased). The size of the window in the unweighed smoothers (see below for the definition of weighted smoothers) plays the role of the *smoothing parameter*. Large widows produce an estimate of the trend that is low in variance but high in bias. Small widows,

conversely, produce an estimate that is high in variance but low in bias. Hence there is always a 'trade-off' between bias and variance. The windows is controlled in the unweighed local regression by the $span = (2k+1)/n$, where k is the number of observations in the left/right of the target (middle) value. The span can take values from 0 to 2. For a very small value close to zero, the window will contain only one observation, while at a value of 2 it will contain all data points. The span for local unweighed polynomial regression is the smoothing parameter.

Definition: A *smoothing parameter* determines how smooth the fitted curve is and it is effectively striking a balance between bias and variance in the estimation the curve. We shall use the Greek letter λ to denote the smoothing parameter in general.

How to choice the smoothing parameter λ is one of the most important topics in the literature of smoothing techniques.

In any local regression scatterplot smoother there are three main decisions that need to be made: i) the size of the window (i.e. the choice of the smoothing smoothing parameter) ii) the degree of polynomial and iii) how the response values are averaged. The second is dealt by fitting different degree polynomial functions in x to the data. The third by deciding whether to used unweighed or weighted polynomial regression. For weighted local regression the specification of a *kernel* function and its smoothing parameter λ is required. Kernels are positive symmetric functions, looking usually similar to the normal distribution, having as a smoothing parameter a scaling parameter which make the shape of the function narrower or wider. For example, if the normal distribution is chosen as kernel, then the the standard deviation $\lambda = \sigma$ is used as a smoothing parameter. It is well known in the smoothing literature that to determine the smoothness of the fitted curve, it is the smoothing parameter λ rather the choice of the kernel that matters. The following describes how the local regression smoothing with a symmetric window works.

- Start by ordering the pair of values (y_i, x_i) for $i = 1, 2, \ldots, n$ with respect to x.
- Use the smoothing parameter to select the size of the window or how wide the kernel function should be.
- Focus on a single observation (with the target x value) and fit a polynomial regression model only to observations falling into the current window (for unweighed local regression) or weight the observations according to the kernel function.
- Use the fitted values $\hat{y}$ of y for the x value of the target observation (x, y) as the fitted value for the smoother (at the target x value).
- Repeat this for all observations.

Figure 9.4 demonstrates some aspect of this process. Each plot shows:

I The current target observation (x, y) in bold and with a pointed arrow,

II The fitted polynomial. For example, Figure 9.4(a) shows a constant fit (or moving average), Figure 9.4(b) a linear fit, Figure 9.4(c) a quadratic fit and finally Figure 9.4(d) a cubic fit.

The plots in Figure 9.4 (a) and (b) are using an unweighed fit and therefore show the chosen windows as shaded areas. Note that the symmetry of the window (that is, containing an equal number k of observations on the left and on the right of the target value) breaks down at the two extreme ends of the range of x. For example, if there are less than k observations on the

left of the target value, the window will contain less observations on the left of the target value than on the right, as demonstrated in Figure 9.4(a). The opposite behaviour will happen in the right part of the data. Note also that a span value close to zero will interpolate the data since there will be left as many observations in the window as the degree of the polynomial. A value of span equal to 2 will fit a global polynomial to all the data, while a value equal to 1 will fit a global polynomial for the middle point of the ordered data but not for the rest of the target points. Usually $span = 0.5$ is a good starting point and this is the value used in Figures 9.4 (a) and (b). The plots in Figure 9.4 (c) and (d) use a weighted fit using a normal kernel with smoothing parameter $\sigma = 0.25$. The shaded area in the plot shows how much weight an observation has in determining the fitted value of y at the target observation. Observations far for the target x value have negligible effect since their weights are close to zero.

Figure 9.4

```
  library(gamlss.demo)
```

```
## Loading required package:  rpanel
## Loading required package:  tcltk
## Package 'rpanel', version 1.1-3:  type help(rpanel) for summary information
```

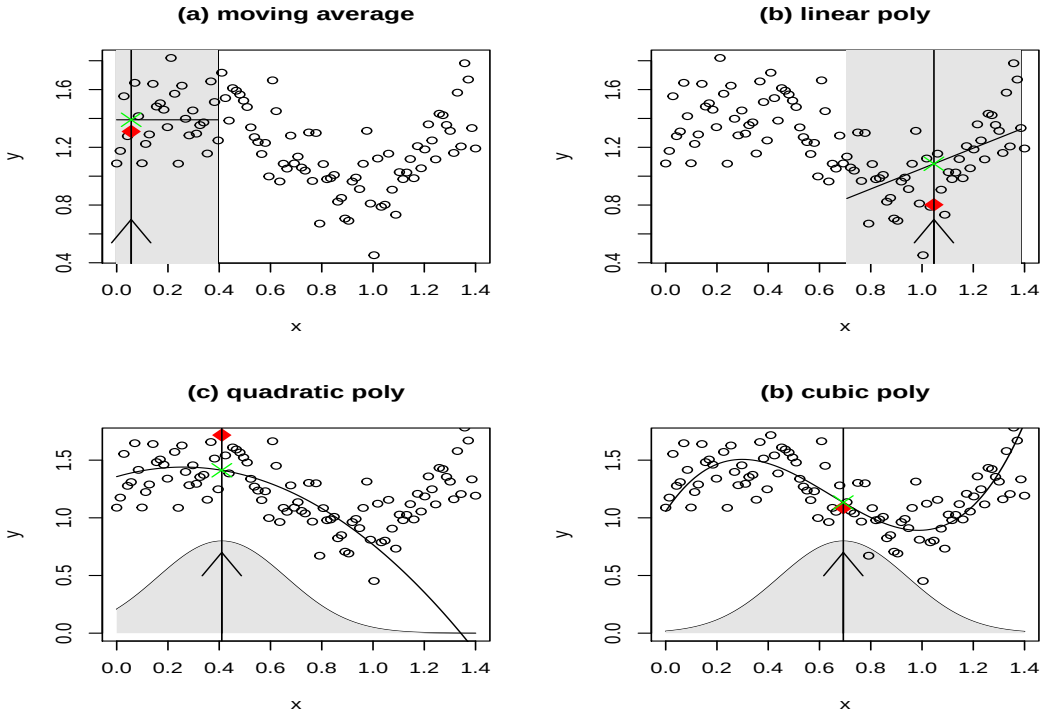
```
n <- 100
x <- seq(0, 1, length = n)*1.4
set.seed(123)
y <- 1.2 + .3*sin(5  * x) + rnorm(n) * 0.2
op <- par(mfrow=c(2,2))
LPOL(y,x, deg=0, position=5)
title("(a) moving average")
LPOL(y,x, deg=1,  position=75)
title("(b) linear poly")
WLPOL(y,x, deg=2, position=30)
title("(c) quadratic poly")
WLPOL(y,x, deg=3, position= 50)
title("(b) cubic poly")
par(op)
```

There are several demos within the package **gamlss.demo** which show how different local regression models work. Readers with less experience in smoothing techniques are encouraged to use them into order to understand the basic ideas behind local regression smoothing. Table 9.1 gives the names of these functions and emphasise their functionality. The demos can be obtained by typing the name of the **R** functions, e.g. `demo.LocPoly()`, or can be accessed using the function `gamlss.demo()` and following the menu for 'Demos for local polynomial smoothing'.

Table 9.1: Showing different ways of using local regression smoothers

	unweighed	weighted
mean (demos)	moving average `demo.LocMean()`	kernel smoothers `demo.WLocMean()`
linear (demos)	simple regression `demo.LocPoly()`	weighted linear regression `demo.WLocPoly()`
polynomial	polynomial regression	weighted polynomial regression

The following are general comments on local polynomial smoothers:



R code on page 215

Figure 9.4: Showing different aspects of fitted local polynomial regression: i) Plots (a) and (b) show unweighed local regression fits with *span* = 0.5 while plots (c) and (d) show a weighted fit using a normal kernel with smoothing parameter $\sigma = 0.25$. Plot (a) uses a constant fit (i.e. a moving average), plot (b) uses a local linear fit, plot (c) a local quadratic fit and plot (d) a local cubic fit.

1. Weighted local polynomial smoothers using a kernel function as weights produce much smoother fits than unweighed local regression using a window. The latter produce rather wiggly functions.
2. The running-mean smoothers (that is, moving average and kernel smoothers shown in the top row of Table 9.1) tend to flatten out the trends at the endpoints of the x variable and thus the fitted values produced are biased at the end points.
3. Every univariate smoother has a smoothing parameter which controls the amount of smoothing done to the data, i.e. the *span* or λ.
4. All the local polynomial regression smoothers discussed up to now are linear in the sense that we can write the vector of fitted values as $\hat{\mathbf{y}} = \mathbf{S}\mathbf{y}$.
5. Smoothers are affected differently when the region of the x values is sparse, for example, in the local weighted window, the values of the weights are affected by the sparseness of the x-values. Also the kernel smoother can misbehave at the end of the range of x-values where data are sparse and more generally when the x-values are unevenly spaced.
6. The smoother takes its values locally since only local observations take part on the fit. As a consequence, the smoother is generally robust to extreme x-values since those values will only contributed locally to the fit. This is contrary to the say polynomial regression fits where extreme x-values have great influence on the whole fitted curve.
7. Influential observations in the y-axis do effect the smoothers. That is why, for example, one of the most successful algorithms for weighted local regression, the `loess` algorithm, provides also a robust version. The implementation of the `loess` function within GAMLSS is discussed later in Chapter ???? [Influential observations in the y-axis are dealt in GAMLSS by fitting robust distributions to the data.]

9.4 Penalised smoothers: univariate.

The penalised smoothers are the most important smoothers within the GAMLSS family of smoothers because of their flexibility and the fact that they can be applied in a variety of different situations. All of the smoothers considered in this section can be thought of as the solution to the following least squares minimisation problem, where certain quadratic constraints apply to the parameters.

Let $\mathbf{Z}$ be a $n \times p$ basis matrix (bases are defined in Chapter **??**), $\boldsymbol{\gamma}$ a $p \times 1$ vector of parameters, $\mathbf{W}$ a $n \times n$ diagonal matrix with weights, $\mathbf{G}$ an $p \times p$ penalised matrix, λ the smoothing parameter and $\mathbf{y}$ the variable of interest. Then penalised smoothers are the solution to the minimisation of the following quantity Q with respect to $\boldsymbol{\gamma}$:

$$Q = (\mathbf{y} - \mathbf{Z}\boldsymbol{\gamma})^\top \mathbf{W}(\mathbf{y} - \mathbf{Z}) + \lambda \boldsymbol{\gamma}^\top \mathbf{G}\boldsymbol{\gamma}. \tag{9.2}$$

The solution to the minimisation problem in equation (9.2) is:

$$\hat{\boldsymbol{\gamma}} = \left(\mathbf{Z}^\top \mathbf{W}\mathbf{Z} + \lambda \mathbf{G}\right)^{-1} \mathbf{Z}^\top \mathbf{W}\mathbf{y}. \tag{9.3}$$

Different $\mathbf{Z}$'s and $\mathbf{G}$'s will produce different smoothers (as we will try to explain in this section), while within GAMLSS different $\mathbf{W}$ are used iteratively within the backfitting algorithm of

GAMLSS (to fit different distributions for the response variable). The fitted values for y are then given by:

$$\begin{aligned} \hat{\mathbf{y}} &= \mathbf{Z}\left(\mathbf{Z}^\top \mathbf{W}\mathbf{Z} + \lambda \mathbf{G}\right)^{-\mathbf{1}} \mathbf{Z}^\top \mathbf{W}\mathbf{y} \\ &= \mathbf{S}\mathbf{y} \end{aligned} \tag{9.4}$$

where $\mathbf{S}$ is the smoothing matrix which plays the same role as the hat matrix plays in least square estimation. Another quantity of interest within GAMLSS is the trace of $\mathbf{S}$ since it is used as the effective degrees of freedom of the smoother,

$$\begin{aligned} tr(\mathbf{S}) &= tr\left[\mathbf{Z}\left(\mathbf{Z}^\top \mathbf{W}\mathbf{Z} + \lambda \mathbf{G}\right)^{-\mathbf{1}} \mathbf{Z}^\top \mathbf{W}\right] \\ &= tr\left[\left(\mathbf{Z}^\top \mathbf{W}\mathbf{Z} + \lambda \mathbf{G}\right)^{-1} \mathbf{Z}^\top \mathbf{W}\mathbf{Z}\right] \end{aligned} \tag{9.5}$$

The form of the penalty matrix $\mathbf{G}$ is also of great interest. Very often is defined as $\mathbf{G} = \mathbf{D}_\mathbf{k}^\top \mathbf{D}_\mathbf{k}$, where the matrix $\mathbf{D_k}$ is a difference matrix of order k. The $\mathbf{D_1}$ and $\mathbf{D_2}$ matrices of order 1 and respectively look like:

$$\mathbf{D}_1 = \begin{bmatrix} -1 & 1 & 0 & \dots & 0 \\ 0 & -1 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & -1 & 1 \end{bmatrix}$$

and

$$\mathbf{D}_2 = \begin{bmatrix} -1 & -2 & 1 & 0 & \dots & 0 \\ 0 & 1 & -2 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & -2 & 1 \end{bmatrix}.$$

Those matrices can be generated easily in **R** using the `diff()` function as the following code shows (no output is given).

```
D1 <- diff(diag(10), diff=1)
D1
D2 <- diff(diag(10), diff=2)
D2
t(D1)%*%D1
t(D2)%*%D2
```

An important feature of the order k is that it introduces a different type of stochastic dependency for the coefficients $\boldsymbol{\gamma}$. For example, $k = 0$ treats the $\boldsymbol{\gamma}$ as a random effects, $k = 1$ as a random walk of order 1, $k = 2$ as random walks of order 2 and so on.

For people familiar with simple least square estimation is worth point out that the penalised least square can be solved by expanding the original data and them use standard least squares software to do the analysis. This can be demonstrated as follows.

Let $\tilde{\mathbf{y}} = \begin{pmatrix} \mathbf{y} \\ \mathbf{0} \end{pmatrix}$, $\tilde{\mathbf{X}} = \begin{pmatrix} \mathbf{Z} \\ \sqrt{\lambda}\mathbf{D} \end{pmatrix}$ and $\tilde{\mathbf{W}} = \begin{pmatrix} \mathbf{W} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_p \end{pmatrix}$ where $\mathbf{0}$ in $\tilde{\mathbf{y}}$ is of length p. Then it is easy to so that minimising the quantity $Q_1 = \left(\tilde{\mathbf{y}} - \tilde{\mathbf{X}}\boldsymbol{\gamma}\right)^\top \tilde{\mathbf{W}} \left(\tilde{\mathbf{y}} - \tilde{\mathbf{X}}\boldsymbol{\gamma}\right)$ leads to the same solution

as in Equation 9.3, $\hat{\boldsymbol{\gamma}} = \left(\mathbf{Z}^\top \mathbf{W} \mathbf{Z} + \lambda \mathbf{G}\right)^{-1} \mathbf{Z}^\top \mathbf{W} \mathbf{y}$. since

$$\left(\tilde{\mathbf{X}}^\top \tilde{\mathbf{W}} \tilde{\mathbf{X}}\right) = \begin{pmatrix} \mathbf{Z} & \sqrt{\lambda}\mathbf{D} \end{pmatrix} \begin{pmatrix} \mathbf{W} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_p \end{pmatrix} \begin{pmatrix} \mathbf{Z} \\ \sqrt{\lambda}\mathbf{D} \end{pmatrix} = \mathbf{Z}^\top \mathbf{W} \mathbf{Z} + \lambda \mathbf{D}^\top \mathbf{D}$$

and

$$\left(\tilde{\mathbf{X}}^\top \tilde{\mathbf{W}} \tilde{\mathbf{y}}\right) = \begin{pmatrix} \mathbf{Z} & \sqrt{\lambda}\mathbf{D} \end{pmatrix} \begin{pmatrix} \mathbf{W} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_p \end{pmatrix} \begin{pmatrix} \mathbf{y} \\ \mathbf{0} \end{pmatrix} = \mathbf{Z} \mathbf{W} \mathbf{y}$$

.

There are several ways of estimating the smoothing parameter within GAMLSS and they are described in Section 11.2. The methods used here are *local* and they are:

- Generalised cross validation (GCV), Wood [2006].
- GAIC ,
- Maximum likelihood method.

This needs to be connected with the chapter in model selection

9.4.1 Demos on penalised smoothers

There are several demos within the package **gamlmss.demo** for helping to understand how penalised smoothers works. They can be accessed by typing `gamlss.demos()` and then clicking on the menu "Demos for smoothing techniques" or by typing the names of the functions below:

demo.BSplines(): This function is designed for exploring the B-splines basis ideas. The user can control the amount of knots for the basis and also the degree of the B-spline. The demo also shows, by clicking the button "random", different shapes of curves than can be generated from a linear combination of such a B-spline basis.

demo.RandomWalk() : This function demonstrates the most basic penalised smoother, the random walk. Random walks are appropriate for time series data when the observations are defined at equal space in time and there is no explicit explanatory variable. It can be seen as the solution of the problem of minimising the quantity Q with respect to μ in $Q = (\mathbf{y} - \boldsymbol{\mu})^\top (\mathbf{y} - \boldsymbol{\mu}) + \lambda \boldsymbol{\mu} \mathbf{D}^\top \mathbf{D} \boldsymbol{\mu}$. The solution is $\hat{\boldsymbol{\mu}} = \left(I + \lambda \mathbf{D}^\top \mathbf{D}\right)^{-1} \mathbf{y}$ where $\mathbf{D}$ is usually a difference matrix of order 1. These smoothers are also called the Whittaker smoothers, Whittaker [1922], Eilers [2003].

demo.interpolateSmo(): This function explores how the fitted values of a random walk behave in the case of *interpolation* and *extrapolation*, that is when data are missing at time points or when we are trying to predict outside the current values of time respectively. The user will find that the interpolation is done by a polynomial of degree $2d-1$, while extrapolation is done by a polynomial of degree $d-1$. Both are done by introducing extra data (at the missing or extrapolation time points) with weights zero.

demo.histSmo(): This function shows the power of penalties when we are trying to smooth a histogram. It is using an old trick within the GLM literature of treating the counts within the histogram bin of as Poisson distributed observations, Eilers and Marx [2010] indexsmoothers!penalised!demo!histogram

demo.PSplines() This demo shows the effect on the fitted P-spline curve of changing i) the number of knots in the B-spline basis, ii) the degrees of the polynomial used iii) the order k of the penalty matrix $\mathbf{D}_k$ and vi) more importantly the smoothing parameter λ.

Next we will consider all univariate penalised smoothers implemented with GAMLSS.

9.4.2 The `pb()`, `pbo()` and the `ps()` functions for fitting a P-splines smoother

The `pb()` stands for Penalised B-splines and it is an implementation in GAMLSS of the Eilers and Marx (1996) P-splines methodology. The functions `pb()` and `pbo()` give identical results but `pb()` is faster than the earlier version which is now under he name `pbo()`. P-splines uses $\mathbf{Z} = \mathbf{B}$ in equation (9.2) where $\mathbf{B}$ is a B-spline basis of a piecewise polynomial of degree d with equal spaced knots over the x range. The coefficients $\boldsymbol{\gamma}$ are penalised using the penalty matrix $\mathbf{G} = \mathbf{D}_\mathbf{k}^\top \mathbf{D}_\mathbf{k}$ of appropriate order k.

The function `pb()` has only three arguments, while the rest can be passed through the `control` option:

`x` The single explanatory variable.

`df` The desired effective degrees of freedom (trace of the smoother matrix minus two for the constant and linear part). This does not need to be an integer but must be positive.

`lambda` the smoothing parameter.

`control` the function `pb.control()` which sets the smoothing parameters.

If both `df` and `lambda` are set to `NULL`, then the smoothing parameter is estimated using one of the methods described below. If `df` is set, then the smoother will have fixed degrees of freedom, `df`. If `lambda` is set, then its value is used for smoothing. If both `df` and `lambda` are set, the `lambda` takes priority.

The `pb.control()` function has the following options:

`inter` the number of equal spaced intervals in x to be used as knots for the creation of the B-splines basis $\mathbf{B}$. The default value os 20.

`degree` the degree of the piecewise polynomial used for the basis $\mathbf{B}$. The default is 3.

`order` the required difference k in the difference matrix $\mathbf{D}_\mathbf{k}$ with default 2.

`start` the starting value for the smoothing parameters `lambda` if it is estimated.

`quantiles` if `TRUE` the quantile values in x are use to determine the knots rather than equally spaced knots.

`method` The method used in the (local) estimation of the smoothing parameters. Available methods are `"ML"`, `"GAIC"` and `"GCV"`. The older version `pbo()` has in addition the methods `"ML-1"`, `"EM"`. The `"ML"` method is described in Rigby and Stasinopoulos [2013]. The `"ML-1"` is an experimental method identical to `"ML"` with the exception that the σ_e parameter is set to 1. This seems to make the algorithm unstable, so it is not recommended. The `"EM"`

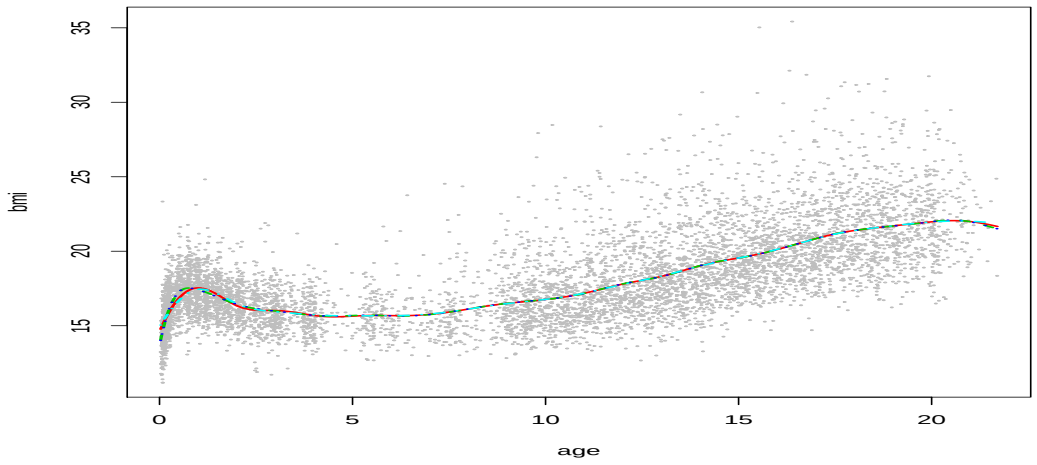
is based on the method described by Fahrmeir and Wagenpfeil [1997], which should give identical results with "ML" but is generally slower.

The `ps()` function is an earlier version of `pb()` and there is no option for estimation of the smoothing parameters. It is based on an original `Splus` function of Brian Marx. The `ps()` uses as default fit a smooth function in x using 3 extra degrees of freedom i.e 5 degrees of freedom overall, 3 for smoothing, one for the linear part and one for the constant.

The following code demonstrates that different methods could lead to slightly different fitted curves. In general for large data use SBC as a smoothing method, for example `method=GAIC)` and `k=log(n)`.

Figure 9.5

```
p1 <- gamlss(bmi~pb(age, method="ML"), data=dbbmi, trace=FALSE)
p2 <- gamlss(bmi~pb(age, method="GCV") , data=dbbmi, trace=FALSE)
p3 <- gamlss(bmi~pb(age, method="GAIC", k=2) , data=dbbmi, trace=FALSE)
p4 <- gamlss(bmi~pb(age, method="GAIC", k=log(length(dbbmi$bmi))),
data=dbbmi, trace=FALSE )
plot(bmi~age, data=dbbmi, cex=.2, col="gray")
lines(fitted(p1)~dbbmi$age, lty=1, col=2, lwd=2)
lines(fitted(p2)~dbbmi$age, lty=2, col=3, lwd=2 )
lines(fitted(p3)~dbbmi$age, lty=3, col=4, lwd=2)
lines(fitted(p4)~dbbmi$age, lty=4, col=5, lwd=2)
```



R code on page 221

Figure 9.5: Different fitted curves using different methods of estimating the smoothing parameters in `pb()`.

Important: For large data use method SBC for a smoother fitted curves.

9.4.3 The pbm() function for fitting a monotonic smooth functions

indexsmoothers!P-splines!monotonic

The function pbm() can be use to fit monotone curves to the data. It is a modified P-splines fit so like the function pb() it uses $\mathbf{Z} = \mathbf{B}$ in equation (9.2) where $\mathbf{B}$ is a B-spline basis of a piecewise polynomial of degree d with equal spaced knots over the x range. The modification is in the penalty part of the fitting The coefficients $\boldsymbol{\gamma}$ are penalised using two penalties matrices: i) one for the smoothness of $\boldsymbol{\gamma}$, $\mathbf{G} = \mathbf{D}_\mathbf{k}^\top \mathbf{D}_\mathbf{k}$ and ii) one which penalised the $\boldsymbol{\gamma}$'s if the monotonic property of the fitted function is violated. The later penalty has the form $\mathbf{P}_k^\top \mathbf{W}_P \mathbf{P}_k$ where the matrix $\mathbf{P}_k$ is defined similar to the penalty matrix $\mathbf{D}_k$ while $\mathbf{W}_P$ is a diagonal matrix of weights which takes the values 1 if the monotonic property of he function is violated and 0 otherwise. Note that the resulted monotonic function is achieved after several iteration of the penalised least square algorithm.

The arguments of the function are almost identical to pb() apart for the argument mono which can take the values "up" (the default) or "down".

Figure 9.6

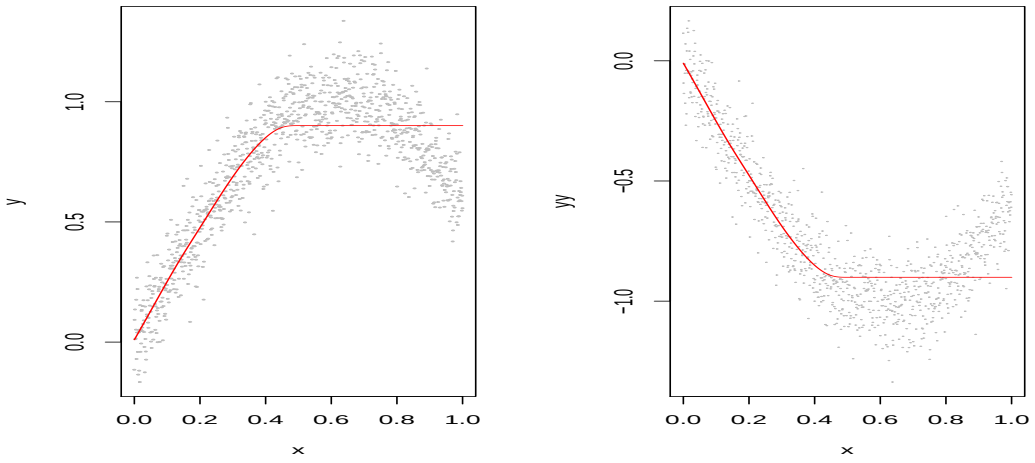
```
  set.seed(1334)
x = seq(0, 1, length = 1000)
p = 0.4
y = sin(2 * pi * p * x) + rnorm(1000) * 0.1
op <- par(mfrow=c(1,2))
plot(y~x, cex=.2, col="gray")
m1 <- gamlss(y~pbm(x), trace=FALSE)
lines(fitted(m1)~x, col="red")
yy <- -y
plot(yy~x, cex=.1, col="gray")
m2 <- gamlss(yy~pbm(x, mono="down"), trace=FALSE)
lines(fitted(m2)~x, col="red")
par(op)
```

9.4.4 The cy() function for fitting a cycle smooth functions

The function cy() produces smooth fitted curves with the property that the two ends (left and right) of the fitted smooth function have identical values. This behaviour is ideal for fitting periodic functions when do not expect the start of a new period to vary considerable from the end of the last one.

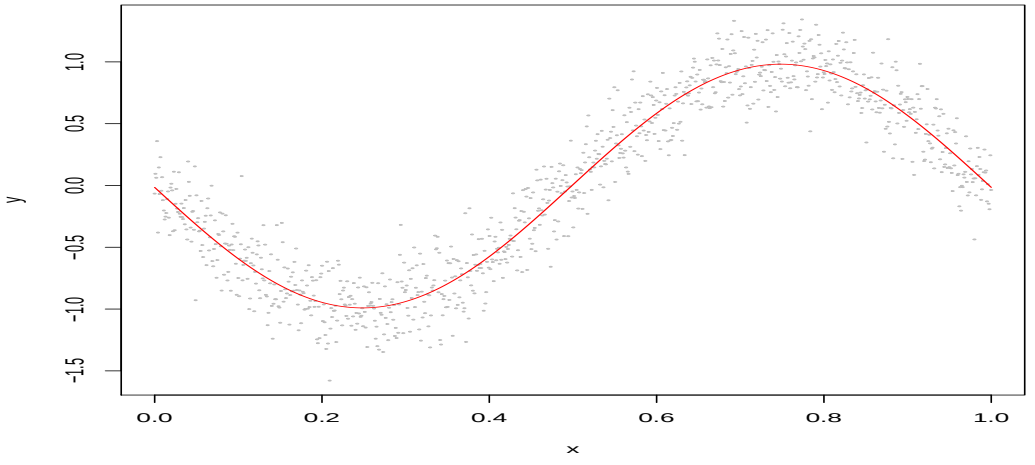
Figure 9.7

```
  set.seed(555)
x = seq(0, 1, length = 1000)
y<-cos(1 * x * 2 * pi + pi / 2)+rnorm(length(x)) * 0.2
plot(y~x, cex=.2, col="gray")
m1<-gamlss(y~cy(x), trace=FALSE)
lines(fitted(m1)~x, col=2)
```



R code on page 222

Figure 9.6: Monotone fitted curves using the function`pbm()`.



R code on page 222

Figure 9.7: Fitted curves ending in the same value they started using the function `cy()`.

9.4.5 The cs() and scs() functions for fitting cubic splines

Both cubic splines functions are based on the `smooth.spline()` function of `R` and can be used for univariate smoothing. They fit a cubic smoothing spline function, see for example Hastie and Tibshirani [1990], Green and Silverman [1994] page 46 or Wood [2006] pages 124 and 149.

The cubic splines are the solution to the following minimisation problem. Let $g(t)$ be a twice differential function of t in the interval $[a, b]$ and λ a smoothing parameter. Define the penalised sum of squares function

$$Q_2(g) = \sum_{i=1}^{n} w_i \left(y_i - g(t_i)\right)^2 + \lambda \int_a^b \left\{g''(x)\right\}^2 dx$$

where w_i for $i = 1 \ldots n$ are prior weights and g'' are the second derivative of the function. It turns out that the minimiser of the function $Q_2(g)$ over the class of all twice-differentiable functions g are cubic splines. Also $Q_2(g)$ can be written as

$$Q_2 = (\mathbf{y} - \mathbf{g})^\top \mathbf{W} (\mathbf{y} - \mathbf{g}) + \lambda \mathbf{g}\mathbf{K}\mathbf{g}$$

for a suitable defined $\mathbf{K}$ matrix, see for details Green and Silverman [1994].

There are two main differences between cubic spline smoothers `cs()` and P-splines `pb()`. The basis function used for cubic smoothing splines fitting is similar to the P-splines. it is a B-spline basis, $\mathbf{B}$ of a piecewise polynomial of degree 3. But while in P-splines we take equal distance knots in the x-axis in cubic smoothing splines the knots are at the distinct x-variable values. The second difference is in the penalty. P-splines achieve smoothness in the fitted function by penalise the parameters $\boldsymbol{\gamma}$. Cubic smoothing splines achieve smoothness by penalised the second derivative of the function. The resulting smoothing curves are usually very similar.

The functions `cs()` and `scs()` behave differently at their default values when the degrees of freedom `df` and `lambda` are not specified. For example `cs(x)` by default will use 3 extra degrees of freedom for smoothing for x (5 all together, if you include the linear and the constant). `scs(x)` by default will estimate lambda (and therefore the degrees of freedom) automatically using generalised cross validation (GCV). Note however that for small data sets the GCV can create instability in the algorithm.

The `cs()` function has the following arguments

`x` the univariate vector of an explanatory variable.

`df` the desired equivalent number of degrees of freedom [trace of the smoother matrix minus two (for the constant and linear fit)]. The real smoothing parameter (spar below) is found such that df=tr(S)-2, where S is the smoother matrix which depends on `spar`. Values for df should be greater than 0, with 0 implying a linear fit. The default is $df = 3$, i.e.. 3 degrees of freedom for smoothing `x` on top of a linear and constant term in `x` giving a total of 5 degrees of freedom.

`spar` smoothing parameter, typically (but not necessarily) in the default range for spar (-1.5,2]. The coefficient lambda of the integral of the squared second derivative in the fitted (penalized log likelihood) criterion is a monotone function of 'spar', see the details in 'smooth.spline' in `R`.

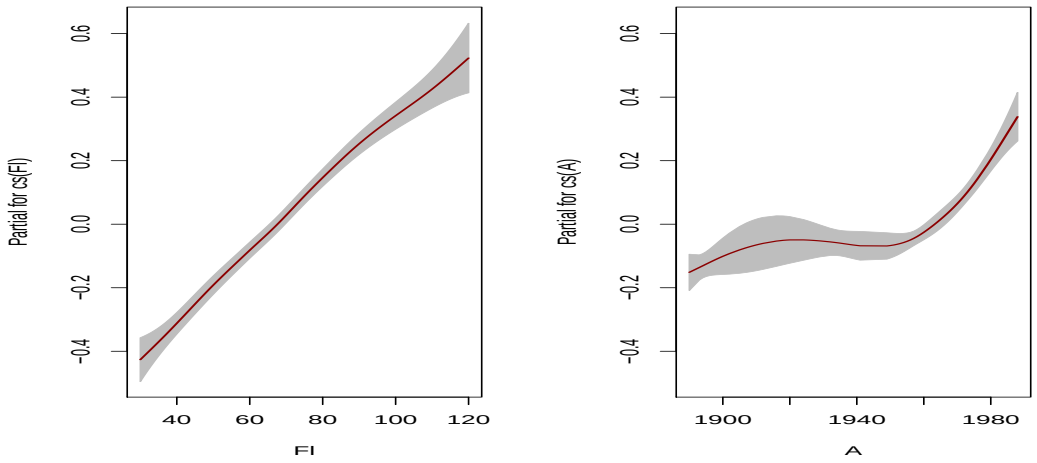
`c.spar` This specifies minimum and maximum limits for the smoothing parameter, the default limits being −1.5 to 2. This is an option to be used when the degrees of freedom of the output fitted `gamlss` object are different from the ones given as input in the option `df`, which is caused by the default limits for the smoothing parameter being too narrow to obtain the required degrees of freedom. The default values used are the ones given the option `control.spar` in the R function 'smooth.spine()' and they are 'c.spar=c(-1.5, 2)'. For very large data sets e.g. 10000 observations, the upper limit may have to increase for example to 'c.spar=c(-1.5, 2.5)'. Use this option if you have received the warning 'The output df are different from the input, change the control.spar'. 'c.spar' can take both vectors or lists of length 2, for example 'c.spar=c(-1.5, 2.5)' or 'c.spar=list(-1.5, 2.5)' would have the same effect.

The `scs()` function has identical arguments plus arguments which can be passed to `smooth.spline()` function.

As an example we used the smoothing cubic spline functions `cs()` and `scs()` to the rent data. We remind the reader that the response variable `R`, is the monthly net rent (rent minus calculated or estimated utility cost). We fit an additive model for floor space `Fl` and the year of construction age `A`.

```
# fitting cubic splines with fixed degrees of freedom
rcs1<-gamlss(R~cs(Fl)+cs(A), data=rent, family=GA, trace=FALSE)
# fitting cubic splines by estimating the smoothing parameter
rcs2<-gamlss(R~scs(Fl)+scs(A), data=rent, family=GA, trace=FALSE)
term.plot(rcs1, pages=1)
```

Figure 9.8



R code on page 225

Figure 9.8: Fitted curves using the function `cs()` (cubic splines).

Next we use `predict` to create the fitted surfaces and plot them as contour plots.

Figure 9.9

```
newrent<-data.frame(expand.grid(Fl=seq(30,120,5),A=seq(1890,1990,5)))
pred1<-predict(rcs1,  newdata=newrent, type="response")

## new prediction

pred2<-predict(rcs1,  newdata=newrent, type="response")

## new prediction

Fln<-seq(30,120,5)
An<-seq(1890,1990,5)
op<-par(mfrow=c(1,2))
contour(Fln,An,matrix(pred1,nrow=length(Fln)),nlevels=30,
ylab="year of construction", xlab="floor space", main="(a)")
contour(Fln,An,matrix(pred2,nrow=length(Fln)),nlevels=30,
ylab="year of construction", xlab="floor space",  main="(b)")
par(op)
```

R code on page 225

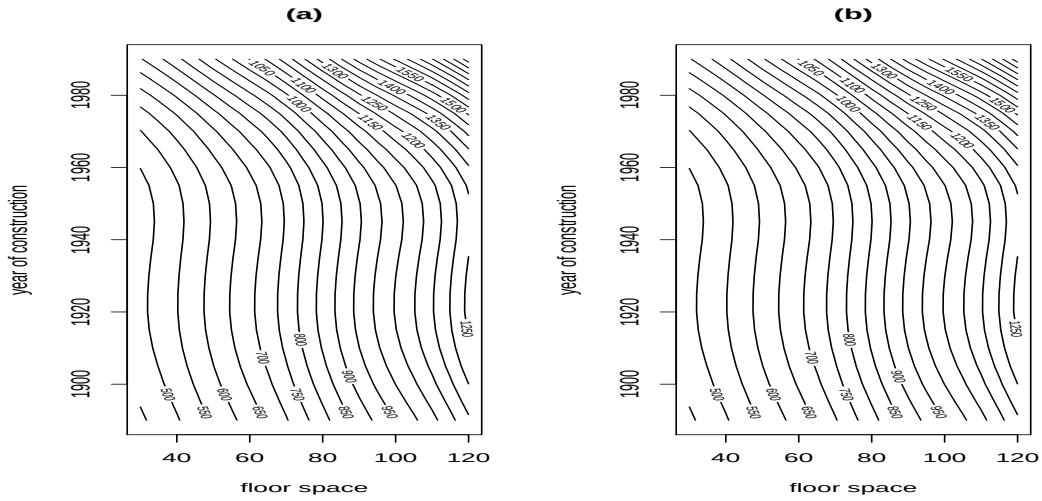
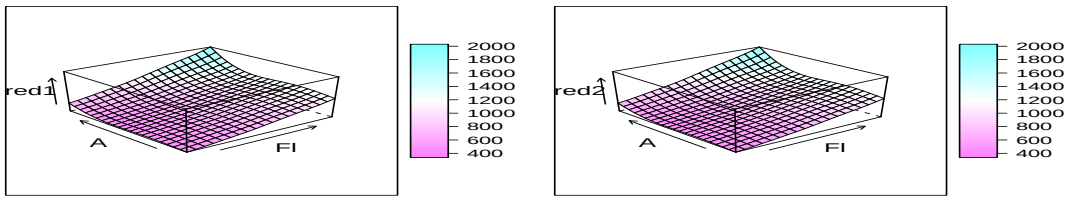


Figure 9.9: Fitted additive curves surface using (a) `cs()` and (b) `scs()` for the rent data. The fitted surfaces are almost identical.

Three dimensional plots of the fitted surfaces can be created as follows:

Figure 9.10

```
library(lattice)
p1<-wireframe(pred1~Fl*A, newrent, aspect=c(1,0.5), drape=TRUE,
colorkey=list(space="right", height=0.6))
p2<-wireframe(pred2~Fl*A, newrent, aspect=c(1,0.5), drape=TRUE,
colorkey=list(space="right", height=0.6))
print(p1, main = "(a)", split = c(1, 1, 2, 1),more = TRUE)
print(p2, main = "(b)", split = c(2, 1, 2, 1))
```



R code on page 226

Figure 9.10: Three dimensional additive surfaces using `cs()` and `scs()` for the rent data.

9.4.6 The `ri()` function for fitting ridge and lasso regression terms

Ridge regression is a simple example of how shrinkage methods can be used for selection of variables. By 'selection of variables' we mean the process in which only a subset of the initial explanatory variables available is selected to put in the final model for a specific parameter of the distribution. Methods on how this can be done in general, are explained in more detail in Chapter **??**. One of the methodologies for selecting explanatory variables which enter linearly in the predictor equation of is the *shrinkage* methods see. Hastie et al (2009) page 61.

The shrinkage methods impose a penalty on the size of the linear coefficients of the explanatory variables. For example, let us consider the simple least squares regression problem where the quantity $(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})^\top(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})$ is minimised with respect to $\boldsymbol{\beta}$, and where the matrix $\mathbf{X}$ contains all the explanatory variables and $\boldsymbol{\beta}$ are the linear coefficients of the regression. The solution to the above least squares problem is given by $\hat{\boldsymbol{\beta}}=(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}$.

Ridge regression coefficients are the solution to the 'penalised' least squares problem

$$(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})^\top(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})+\lambda\boldsymbol{\beta}^\top\boldsymbol{\beta} \tag{9.6}$$

with solution

$$\hat{\boldsymbol{\beta}}=(\mathbf{X}^\top\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{y}. \tag{9.7}$$

The effect of the quadratic penalty $\lambda\boldsymbol{\beta}^\top\boldsymbol{\beta}$ is to shrink the least square coefficients towards zero. It is not difficult to see that equation (9.6) is similar to equation (9.2) with $\mathbf{Z}$, $\boldsymbol{\gamma}$ and $\mathbf{D}$ replaced by with $\mathbf{X}$ and $\boldsymbol{\beta}$ and $\mathbf{I}$ respectively and that equation (9.7) is similar to 9.3.

The penalty in equation (9.6) can be replaced with a more general penalty in the form of:

$$(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})^\top(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})+\lambda|\boldsymbol{\beta}|_p^p \tag{9.8}$$

where $|\boldsymbol{\beta}|_p^p = \sum_J |\beta_J|^p$ and where the summation is over all the elements of $\boldsymbol{\beta}$. Different p values define different *norms* of penalties: e.g. $p = 2$ defines the L_2 norm with a quadratic penalty, $p = 1$ defines the L_1 norm with an absolute value penalty, etc. Note L_1 is known as the *lasso* penalty. Different penalties shrink the least squares coefficients towards zero in different ways. The way least squares coefficients are shrunk towards zero for different penalties is described nicely in Hastie et al. (2009) pages 69-73. The effective degrees of freedom used in the shrinkage is given by the trace of the smoothing matrix.

Ridge regression can be fitted within gamlss using the `ri()` function. The most important arguments of the `ri()` are:

`X`	the design matrix of the explanatory variables $\mathbf{X}$ (whose columns are standardised automatically to mean zero and standard deviation one).
`df, lambda`	effective degrees of freedom and smoothing parameter which act the same way as in all other smoothers
`method`	with only local "ML" and "GAIC" as supporting methods for automatic selection of variables.
`Lp`	the type of shrinkage penalty with `Lp=2`, i.e. ridge regression, as default.

In the example below, we use the `usair` data, which has six explanatory variables `x1:x6` and only 41 observations. First, we create the matrix `X` containing all the explanatory variables and we standardised it using the **R** function `scale()`. We then fit four different models. The first is the least squares model, and the rest are different shrinkage approaches i) ridge ii) lasso and iii) 'best subset'.

```
X<-with(usair, cbind(x1,x2,x3,x4,x5,x6))
# standarise the data
sX<-scale(X)
# least squares
m0<- gamlss(y~sX, data=usair, trace=FALSE)
# ridge
m1<- gamlss(y~ri(sX), data=usair, trace=FALSE)
# lasso
m2<- gamlss(y~ri(sX, Lp=1), data=usair, trace=FALSE)
# best subset
m3<- gamlss(y~ri(sX, Lp=0), data=usair, trace=FALSE)
AIC(m0,m1,m2,m3)
```

```
##          df      AIC
## m2 5.336309 341.2492
## m0 8.000000 344.7232
## m1 5.884452 345.6097
## m3 2.838310 350.1807
```

Note that instead `sX` above we could have given just `X` with the same results as it standardised automatically. Model `m2`, the lasso, seems more appropriate here. The different coefficients of the four fitted models are displayed below.

```
cbind(
zapsmall(coef(m0)[-1], digits=4),
```

```
zapsmall(coef(getSmo(m1)), digits=3),
zapsmall(coef(getSmo(m2)), digits=3),
zapsmall(coef(getSmo(m3)), digits=3))
```

```
##        [,1]  [,2]   [,3]  [,4]
## sXx1  -9.16 -9.44  -8.76 -6.69
## sXx2  36.58 18.54  26.54 13.39
## sXx3 -22.75 -5.43 -12.95  0.00
## sXx4  -4.55 -4.16  -3.94  0.00
## sXx5   6.03  4.71   4.73  0.00
## sXx6  -1.38  0.70   0.00  0.00
```

The ridge regression, `Lp=2`, just shrinks the least squares coefficients towards zero. The lasso, `Lp=1`, does the same, but also sets the coefficient of `x6` to zero. The best subset, `Lp=0`, sets four coefficients to zero leaving only `x1` and `x2`. Next we plotting the fitted coefficients from the three different shrinkage methods.

Figure 9.11

```
library(lattice)
op <- par(mfrow=c(3,1))
plot(getSmo(m1)) #
plot(getSmo(m2))
plot(getSmo(m3))
par(op)
```

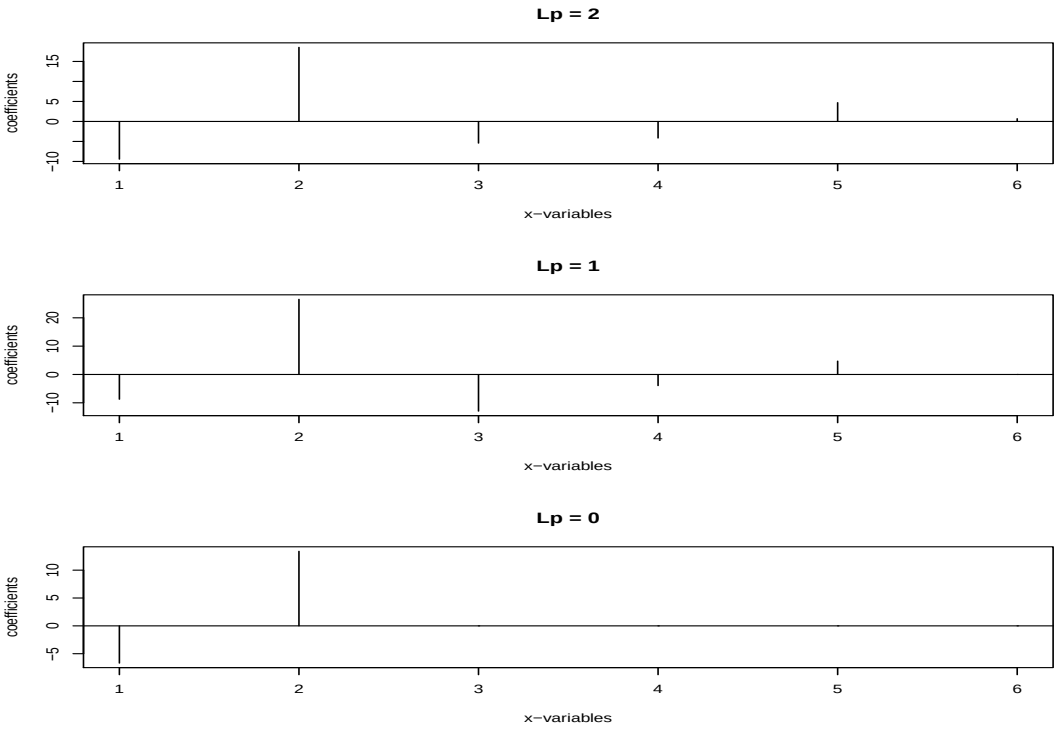
9.5 Penalised smoothers: multivariate

9.5.1 The `pvc()` function for fitting varying coefficient models

The varying coefficient terms were introduced by Hastie and Tibshirani [1993] to accommodate a special type of interaction between explanatory variables. This interaction takes the form of $\gamma(x)z$, Mikis: should I use γ or β here?? that is the linear coefficient of the explanatory variable z is changing smoothly according to another explanatory variable x. In time series data x can be time so the linear relationship between the parameters and x varies over time. More general x should be a continuous variable while z can be either continuous or categorical.

The `pvc()` function has the following arguments

`x`	represents the vector of the explanatory variable x which effects the coefficients of the explanatory variable, `z` i.e. $\gamma(x) * z$.
`df`	as in function `pb()`.
`lambda`	as in function `pb()`.
`by`	the explanatory z variable. [Note that if `z` is continuous variable(rather than a factor) then it is centred automatically i.e $z-\bar{z}$ by `pvc()` due to the invariance of varying coefficient models to location shifts in `z`, see comments of Green in the discussion of Hastie and Tibshirani [1993]
`control`	options for controlling the P-splines fitting.



R code on page 229

Figure 9.11: Plotting the fitted linear coefficients using three different shrinkage approaches: i) ridge (top plot) ii) lasso (middle plot) and iii) best subset (bottom plot).

Using z as continuous variable

As an example of using the function `pvc()` where the `by` argument is a continuous variable, consider the model in which smooth functions for age `A` and floor space `Fl` are fitted but in which we are also like to investigate whether linear or varying coefficients interaction exist between the two variables.

```
# main smoothing effect for Fl and A
m0<-gamlss(R~pb(Fl)+pb(A), data=rent,  family=GA, trace=FALSE)
# linear interaction between A and Fl
m1<-gamlss(R~pb(Fl)+pb(A)+A:Fl,  data=rent,  family=GA, trace=FALSE)
# varying coefficients interaction b(A)Fl
m2<-gamlss(R~pb(Fl)+pb(A)+pvc(A, by=Fl), data=rent,  family=GA, trace=FALSE)
# varying coefficients interaction b(Fl)A
m3<-gamlss(R~pb(Fl)+pb(A)+pvc(Fl, by=A),  data=rent,  family=GA, trace=FALSE)
# linear interaction plus varying coefficients interaction b(A)Fl
m4<-gamlss(R~pb(Fl)+pb(A)+A:Fl+pvc(A, by=Fl),  data=rent,  family=GA, trace=FALSE)
AIC(m0, m1,m2,m3, m4)
```

```
##           df      AIC
## m2 10.925246 27927.36
## m4 11.925247 27929.36
## m1  8.059831 27938.16
## m0  7.371373 27938.35
## m3 10.315233 27938.37
```

Model `m2` with varying coefficient interaction, $\gamma(A)Fl$ seems the more appropriate here. Note however that the ultimatum interaction model is the one which a "smooth" surface fitted for both explanatory terms, as discussed in section 9.5.2. The `term.plot()` function for continuous z in the varying coefficient situation plots the relationship of the estimated beta coefficients $\gamma(x)$ against x as Figure 9.12 is shown.

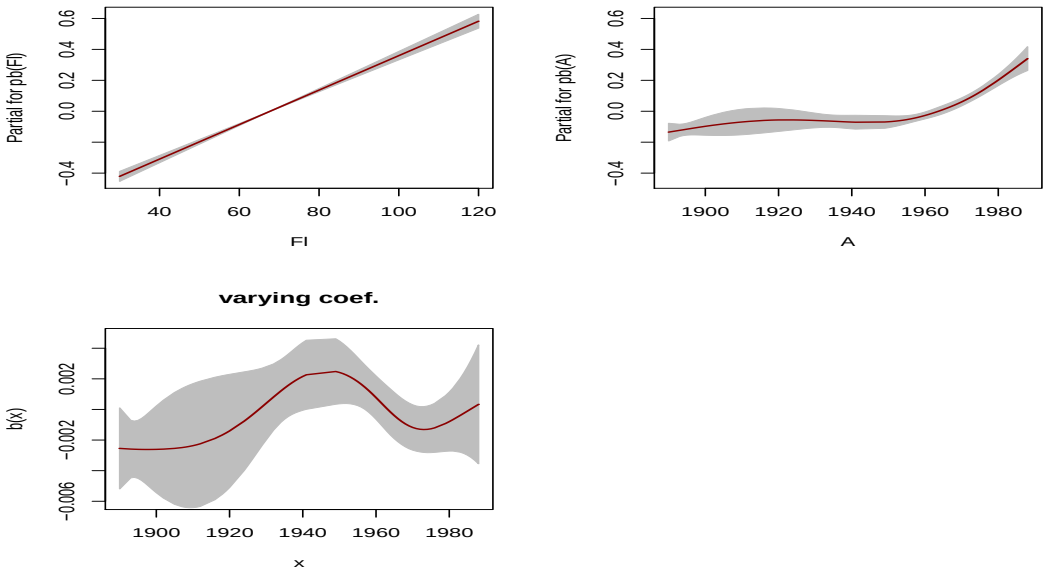
Figure 9.12

```
term.plot(m2, pages=1)
```

The relationship of the estimated beta coefficients $\gamma(x)$ against x of Figure 9.12 does provides some information for on how the γ is changing but as with all two-way interactions of continuous variables a more informative plot is a a contour plot or a three-dimensional of the relationship. In our case since only two explanatory variables are involved a contour plot of the fitted varying coefficient interaction is easy to produce.

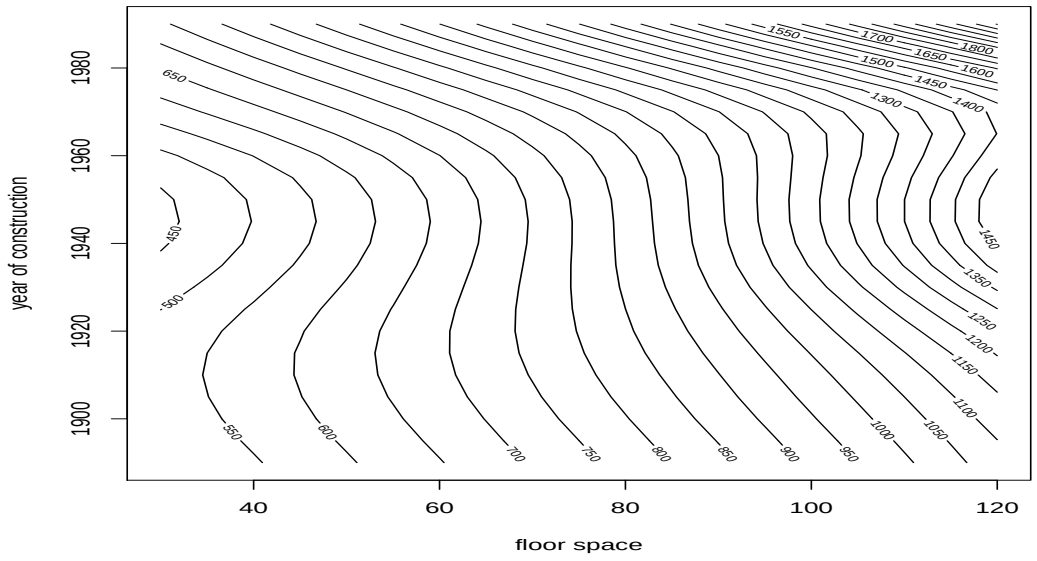
Figure 9.13

```
  newrent<-data.frame(expand.grid(Fl=seq(30,120,5),A=seq(1890,1990,5)))
newrent$pred<-predict(m2,newdata=newrent, type="response", data=rent)

## new prediction

Fln<-seq(30,120,5)
An<-seq(1890,1990,5)
op <- par(mfrow=c(1,1))
contour(Fln,An,matrix(newrent$pred,nrow=length(Fln)),nlevels=30,
        ylab="year of construction", xlab="floor space")
```



R code on page 231

Figure 9.12: The term plot for the varying coefficient interaction model `m2`.



R code on page 231

Figure 9.13: The fitted surface plot of the varying coefficient interaction model `m2`

Using z as factor

When, the z variable used in the argument `by`, is a factor, the varying coefficient function fits separate smooth curves for each level of the factor z against x. In the next example we use the factor `loc`, which identify different locations, (below, average or above average), to demonstrate the point.

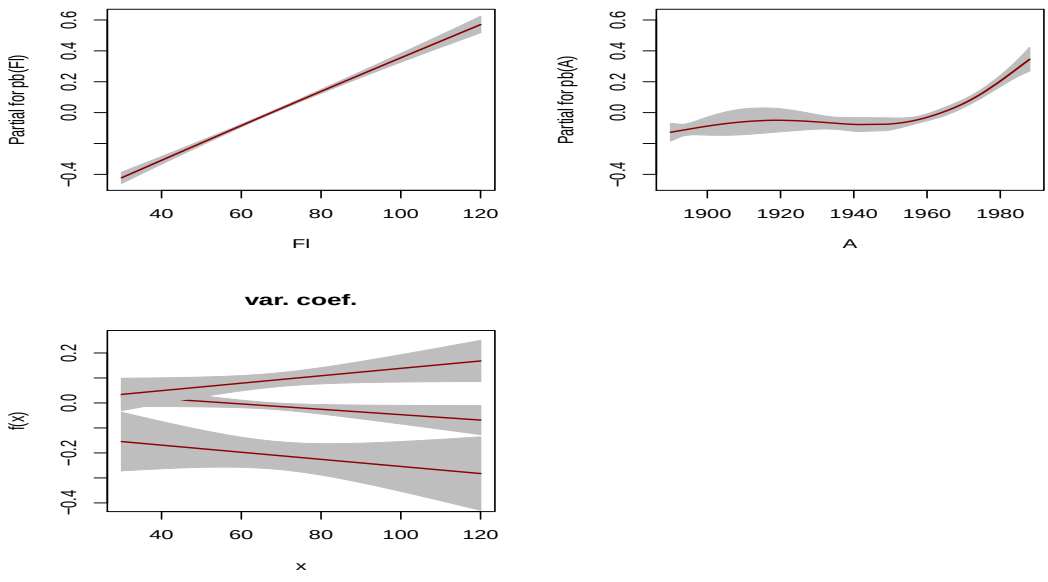
```
g1<-gamlss(R~pb(Fl)+pb(A)+pvc(Fl,by=loc),  data=rent,  family=GA, trace=FALSE)
g2<-gamlss(R~pb(Fl)+pb(A)+pvc(A,by=loc),  data=rent,  family=GA, trace=FALSE)
g3<-gamlss(R~pb(Fl)+pb(A)+pvc(Fl,by=loc)+pvc(A,by=loc),  data=rent, trace=FALSE,
           family=GA)
AIC(g1,g2,g3)
```

```
##           df      AIC
## g1 11.38144 27858.10
## g2 11.97336 27859.57
## g3 15.88057 27859.64
```

Figure 9.14 shows the results of using the `term.plot()` function for model `g1`.

Figure 9.14

```
term.plot(g1, pages=1)
```



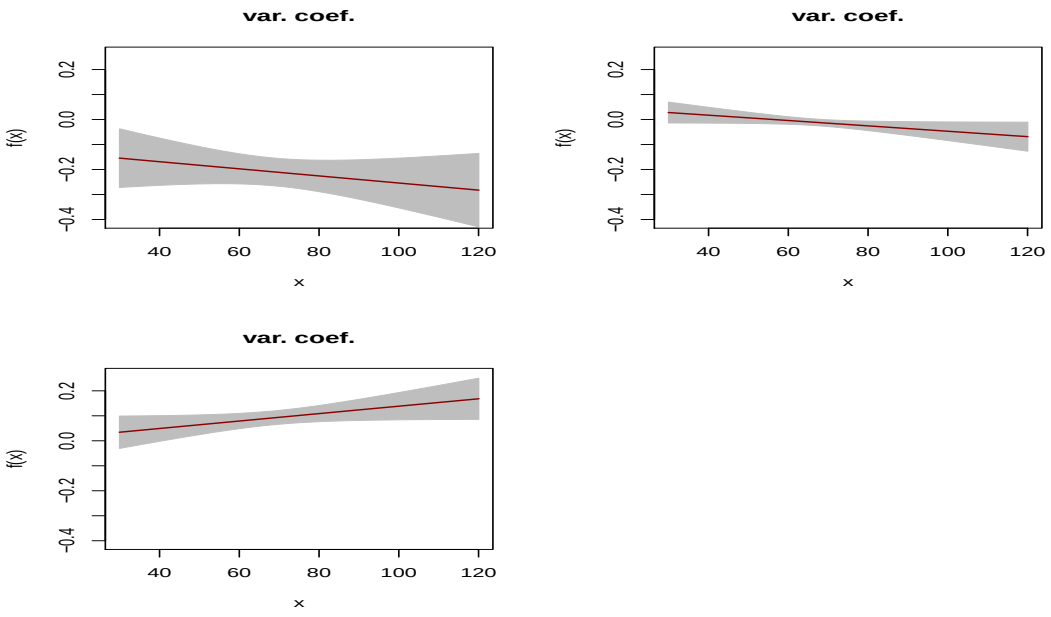
R code on page 233

Figure 9.14: The term plot figures from model `g1`

The three different smooth fits are cramped within the third panel of Figure 9.14. Individual fitted smooth curves can be shown using the following commands:

Figure 9.15

```
op<- par(mfrow=c(2,2))
plot(getSmo(g1, para="mu", which=3),  factor.plots = TRUE)
par(op)
```



R code on page 233

Figure 9.15: Plotting the individual fitted smooth curves from model `g1`

9.5.2 Interfacing with `gam()`, the `ga()` function

The `ga()` function is an additive smoothing function which be used within a GAMLSS models. It is an interface for the `gam()` function of package **mgcv** of Simon Wood. The function can be found in the extra package **gamlss.add** where other interfaces of this kind can be also found. The function `ga()` allows the user to use all the available smoothers of the `gam()` function of the package **mgcv** within the GAMLSS framework. To see which smoothers are available within the package **mgcv** use `?formula.gam`. The great advantage of course of using the smoothers within GAMLSS comes from the fact that this way the fitting models can be outside the exponential family.

For simple one dimensional smoothers using `pb()` or the `ga()` interface make little difference in the resulting fitted smoothing terms. In our experience, for exponential family models, the fitted smoothing curves using a single smoother in `gam()` within **mgcv** or `pb()` within **gamlss** produce very similar results. Therefore the great advantage of the interface function `ga()` is the use of more than one dimensional smoothers like thin plate cubic splines, `s()` or tensor product, `tp()`, which are efficiently implemented within the package **mgcv**.

The function `ga()` has two arguments. The first is a `gam()` type of formula and the second is

the gam() control.

Here we demonstrate the use of the function ga() by fitting different models for floor, Fl, and age of construction, A, to the Munich rent data. Firstly we used smooth additive terms for Fl and A (main effects) and later we fit a smooth surface which explores the interaction between them.

Additive terms

We use the normal and the gamma error distributions as examples. Three different models are fitted using first the function gam() and then using gamlss, calling the interface with gam() and the smoothing function pb() respectively. The resulting deviances and effective degrees of freedom of the fitted models are displayed using GAIC().

```
  library(mgcv); library(gamlss.add)
data(rent)
# additive fits
# normal distribution
ga1 <- gam(R~s(Fl)+s(A), method="REML", data=rent)
gn1 <- gamlss(R~ga(~s(Fl)+s(A), method="REML"),  data=rent, trace=FALSE)
gb1 <- gamlss(R~pb(Fl)+pb(A), data=rent, trace=FALSE) # additive
AIC(ga1, gn1, gb1, k=0)
```

```
##           df      AIC
## ga1 9.258644 28264.38
## gn1 8.351358 28264.38
## gb1 8.372701 28264.19
```

```
# gamma distribution
ga2 <- gam(R~s(Fl)+s(A), method="REML", data=rent, family=Gamma(log))
gn2 <- gamlss(R~ga(~s(Fl)+s(A), method="REML"), data=rent, family=GA,
              trace=FALSE)
gb2 <- gamlss(R~pb(Fl)+pb(A), data=rent, family=GA, trace=FALSE)
AIC(ga2, gn2, gb2, k=0)
```

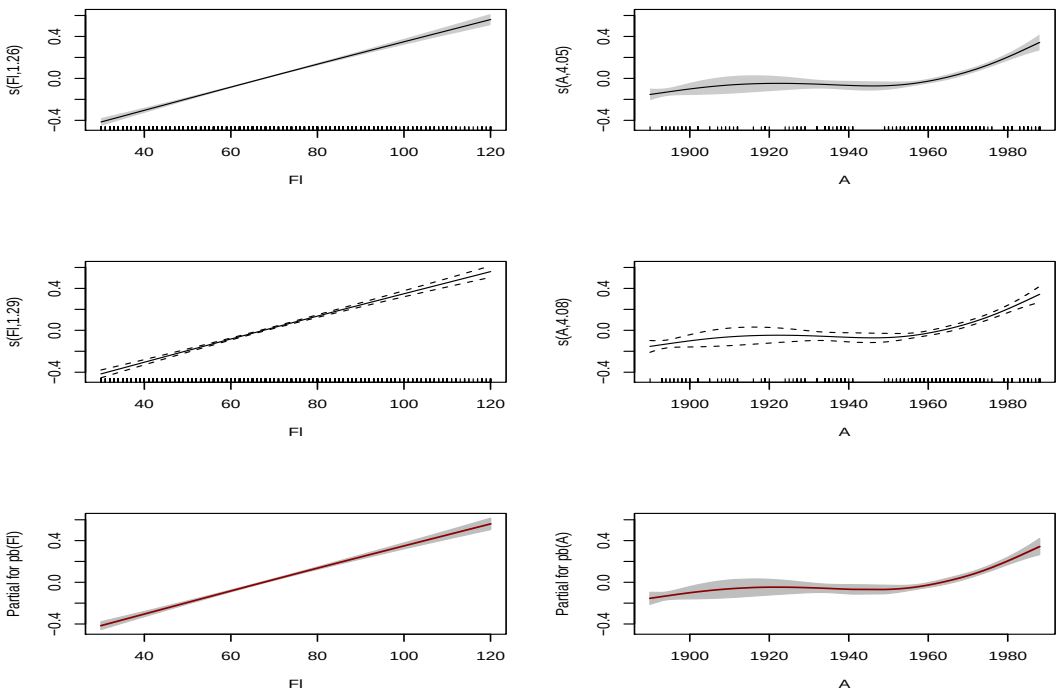
```
##           df      AIC
## ga2 8.295446 27924.42
## gn2 7.370424 27923.69
## gb2 7.371373 27923.61
```

For the normal errors model the fitted deviance are identical but with slightly different degrees of freedom for the gam() model. For the gamma error model the gamlss() function models give almost identical results but slightly different to the gam() model.

Figure 9.16 shows the three resulting plots of the fitted terms for the different models using the gamma distribution for the response. The top rows shows the model fitted using gam(), the middle shows the model fitted using gam() within gamlss() while the bottom row shows the gamlss() using the pb() function. For all practical purposes the three plots leads to identical conclusions.

Figure 9.16

```
  op<-par(mfrow=c(3,2))
plot(ga2, scheme=1)
term.plot(gn2)
term.plot(gb2)
par(op)
```



R code on page 235

Figure 9.16: The plotting of terms of a Gamma distribution models fitted using alternative methods: i) Top rows: using `gam()` ii) Middle row: using `gam()` within `gamlss()` and iii) bottom row: Using `pb()` within `gamlss()`.

Smooth surface fitting

For surface fitting the package `mgcv` provides several options.For example, `s()` for thin plate splines, `te()` for tensor products, `ti()` which is a variant of tensor product designed to be used for interaction terms when the main effects (and any lower order interactions) are present. Next we use thin plate splines:

Figure 9.17

```
  ga4 <-gam(R~s(Fl,A), method="REML", data=rent, family=Gamma(log))
gn4 <- gamlss(R~ga(~s(Fl,A), method="REML"), data=rent, family=GA)

## GAMLSS-RS iteration 1: Global Deviance = 27892.31
## GAMLSS-RS iteration 2: Global Deviance = 27892.34
```

```
## GAMLSS-RS iteration 3: Global Deviance = 27892.34
```

```
AIC(ga4,gn4, k=0)
```

```
##           df      AIC
## ga4 17.43153 27893.40
## gn4 17.10712 27892.34
```

```
op<-par(mfrow=c(1,2))
vis.gam(ga4)
vis.gam(getSmo(gn4))
par(op)
```

R code on page 236

Figure 9.17: Surface fitting of the Gamma distribution models fitted using: i) left: `gam()` ii) right: `gam()` within `gamlss()` .

Note that `term.plot()` will produce a contour plot similar to the `plot(getSmo(gn4))` command.

Figure 9.18

```
  term.plot(gn4)
```

For tensor products smoothers the `gam()` function `te()` can be used.

Figure 9.19

```
  ga5 <- gam(R~te(Fl,A), data=rent, family=Gamma(log))
gn5 <- gamlss(R~ga(~te(Fl,A)), data=rent, family=GA)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 27887.83
## GAMLSS-RS iteration 2: Global Deviance = 27887.83
```

```
AIC(ga5,gn5, k=0)
```

R code on page 237

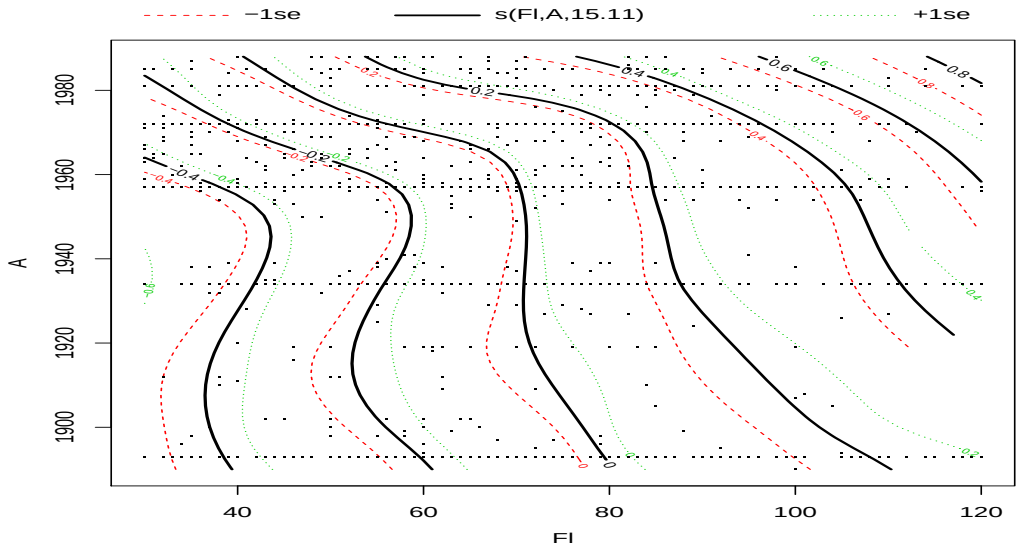


Figure 9.18: Contour plot for a `gam()` model fitted within `gamlss()`.

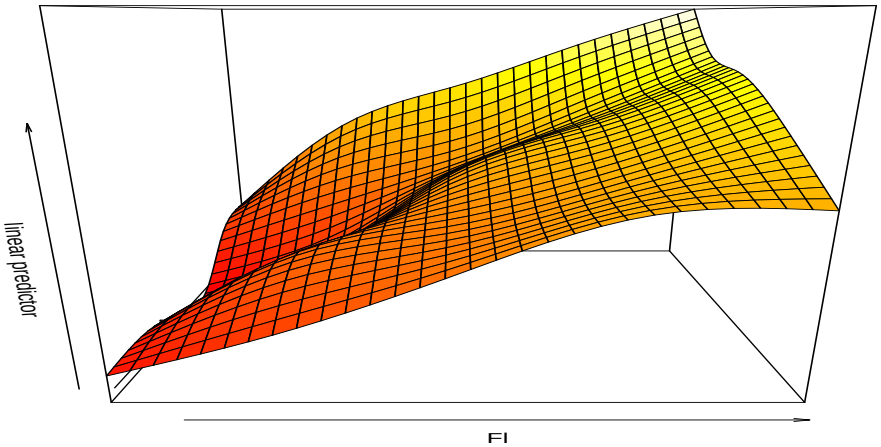
```
##           df      AIC
## ga5 18.14628 27889.38
## gn5 18.61222 27887.83

vis.gam(getSmo(gn5))
```

9.6 Other smoothers

9.6.1 Interfacing with `nnet()`, the `nn()` function

Neural networks provides a flexible way of fitting non-linear regression models, see Bishop et al. [1995] and Ripley [1993, 1996]. They are over-parametrised non-linear statistical models, a fact which allows them to be very flexible and therefore can approximate any smooth function. Because of the over-parametrisation nature of the neural network, they are very difficult model to interpreted compared to the more traditional smoothing models. They are typical '*black box*' models, a name refer to models who work in practice but difficult to show why. On the other hand, they can pick up hight level interactions among the explanatory variables that are very difficult to find otherwise through the more classical regression approach.

The package `gamlss.add` provides an interface with the standard **R** function `nnet()` from the package **nnet**. More information about the use of the `nnet()` function can be found at Venables and Ripley [2002]. The GAMLSS interface is called `nn()` and it takes as arguments a formula plus other control arguments needed for `nnet()`.



R code on page 237

Figure 9.19: Contour plot for a `gam()` model fitted within `gamlss()`.

Note that because the neural network models are over-parametrised different initial values can result to different final fitted model. This is more apparent within a GAMLSS model where the `nnet()` function is called repetitively within the backfitting algorithm. Setting the random generating seeds in the beginning of the GAMLSS fit will insured that the same model can be repeated later. Also one way to help the optimisation precess and possibly to avoid over-fitting is the use of the argument `decay`. `decay` is a smoothing parameter within `nnet`.

Here we demonstrate the use of the `nn()` function by fitting three different models: Firstly, we fit a surface models for floor, `Fl`, and age of construction, `A`, to the Munich rent data. In the second model we add the main effects of the factors `B`, whether there is a bathroom, `H`, whether there is a central heating, `L` whether the kitchen equipment is above average, and `loc` for three different locations. In the last model and in order to explore the interactions facilities of the neural network we fit a neural network model with all explanatory continuous and categorical variables.

```
library(gamlss.add)
set.seed(1432)
mr1 <- gamlss(R~nn(~Fl+A, size=5, decay=0.01), data=rent, family=GA, )

## GAMLSS-RS iteration 1: Global Deviance = 27961.17
## GAMLSS-RS iteration 2: Global Deviance = 27961.17

mr2 <- gamlss(R~nn(~Fl+A, size=5, decay=0.01) +H+B+loc, data=rent, family=GA)

## GAMLSS-RS iteration 1: Global Deviance = 27737.82
## GAMLSS-RS iteration 2: Global Deviance = 27737.82
```

```
mr3 <- gamlss(R~nn(~Fl+A+H+B+loc, size=5, decay=0.01), data=rent, family=GA)

## GAMLSS-RS iteration 1: Global Deviance = 27700.95
## GAMLSS-RS iteration 2: Global Deviance = 27700.95

AIC(mr1, mr2, mr3)

##     df      AIC
## mr3 43 27786.95
## mr2 27 27791.82
## mr1 23 28007.17

AIC(mr1, mr2, mr3, k=log(1969))

##     df      AIC
## mr2 27 27942.62
## mr3 43 28027.12
## mr1 23 28135.63
```

The AIC select the `mr3` model while BIC select `mr2`. The `side` argument is the number of hidden layer of the neural network. The `nnet` fitted object can be retrieved using the function `getSmo()`. For example to get the fitted coefficients use:

```
summary(getSmo(mr3))

## a 6-5-1 network with 41 weights
## options were - linear output units  decay=0.01
##  b->h1 i1->h1 i2->h1 i3->h1 i4->h1 i5->h1 i6->h1
##  -0.65  -0.04   0.00   0.62   0.27   1.55  -3.82
##  b->h2 i1->h2 i2->h2 i3->h2 i4->h2 i5->h2 i6->h2
##   0.13   1.91  -0.11   0.75   0.00   0.75  -0.44
##  b->h3 i1->h3 i2->h3 i3->h3 i4->h3 i5->h3 i6->h3
##   0.72  -0.47   0.03   2.03   0.00  -0.08   0.97
##  b->h4 i1->h4 i2->h4 i3->h4 i4->h4 i5->h4 i6->h4
##  -3.56   0.01   0.00  -0.28  -0.35   0.29  -0.24
##  b->h5 i1->h5 i2->h5 i3->h5 i4->h5 i5->h5 i6->h5
##  -0.13  -0.86   0.05   0.84   0.00  -1.43   2.07
##  b->o h1->o h2->o h3->o h4->o h5->o
## -0.28 -0.68 -0.24 -2.39  3.64  1.33
```

Figure 9.20

A visual presentation of the fitted neural network model can be obtained as follows:

```
plot(getSmo(mr3), y.lab=expression(g[1](mu)))
```

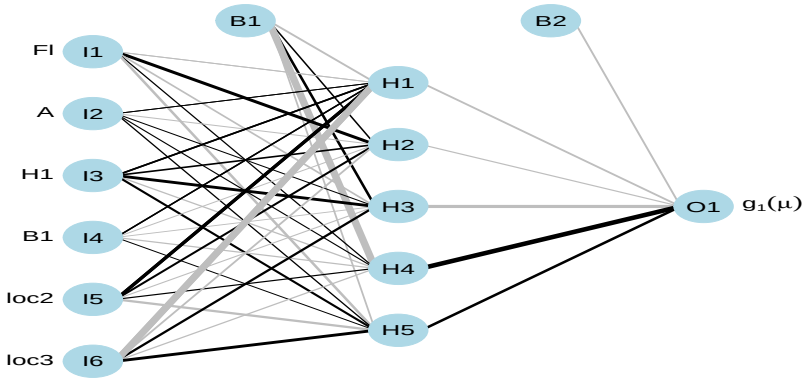
Thinker lines represent coefficients with high values while 'black' and 'grey' colours represent positive and negative values respectively.

Figure 9.21

The function `term.plot()` works in general with `nn()` and it produces a sensible output as the following plot for model `mr2` shows:

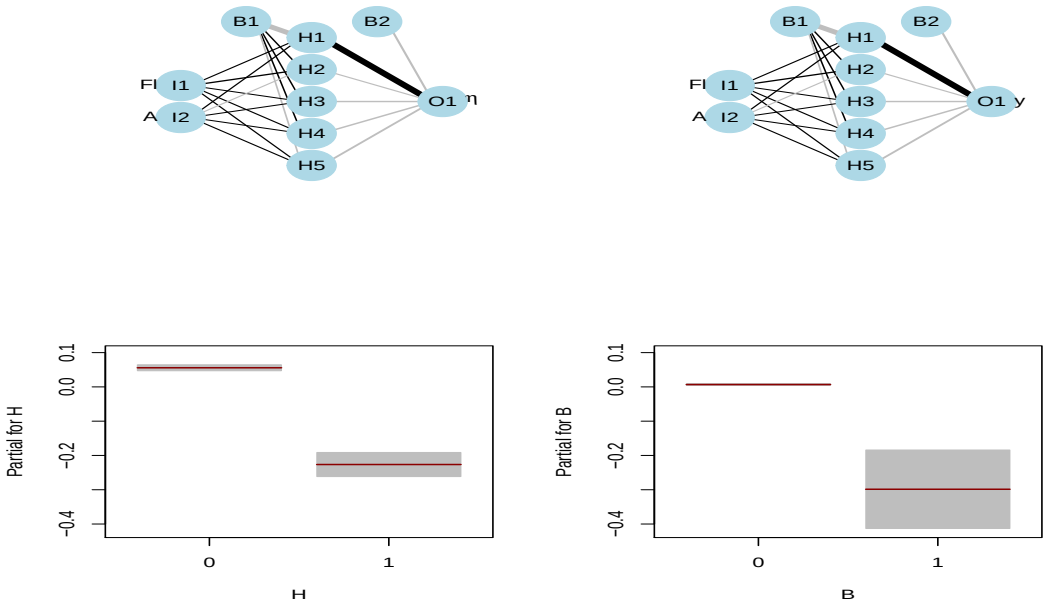
```
term.plot(mr2, pages=1)
```

Next we fit a neural network model to the σ parameter of the gamma distributions and compare the model with the previously fitted models where only μ was modelled as a function of the



R code on page 240

Figure 9.20: Visual representation of the neural network model fitted for μ in model `mr3`.



R code on page 240

Figure 9.21: Visual representation of the neural network model fitted for μ in model `mr3`.

explanatory variables.

```
mr4 <- gamlss(R~nn(~Fl+A+H+B+loc, size=5, decay=0.1),
sigma.fo=~nn(~Fl+A+H+B+loc, size=5, decay=0.1),data=rent,
family=GA, gd.tol=1000)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 27585.52
## GAMLSS-RS iteration 2: Global Deviance = 27534.82
## GAMLSS-RS iteration 3: Global Deviance = 27529.57
## GAMLSS-RS iteration 4: Global Deviance = 27529.57
## GAMLSS-RS iteration 5: Global Deviance = 27529.57
```

```
AIC(mr2, mr3, mr4)
```

```
##     df      AIC
## mr4 84 27697.57
## mr3 43 27786.95
## mr2 27 27791.82
```

```
AIC(mr2, mr3, mr4, k=log(1969))
```

```
##     df      AIC
## mr2 27 27942.62
## mr3 43 28027.12
## mr4 84 28166.73
```

Again the AIC favours the more complicated model `mr4` while the BIC the simpler model `mr2`.

The graphical representation of the fitted `mr4` model is displayed below:

Figure 9.22

```
par(mfrow=c(2,1))
plot(getSmo(mr4), y.lab=expression(g[1](mu)))
plot(getSmo(mr4, what="sigma"), y.lab=expression(g[2](sigma)))
par(op)
```

9.6.2 Interfacing with `rpart()`, the `tr()` function

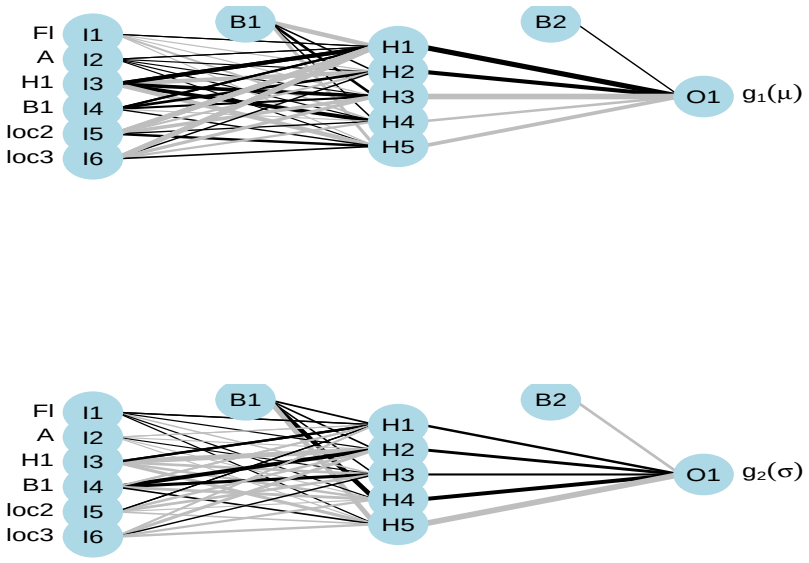
The function `tr()` provides an interface for the `rpart()` function of package **rpart**. This way, decision trees can be fitted as additive terms within GAMLSS. Here we give a small example on how this function can be used. Note however that the function, is rather experimental and of no unique solution is guaranteed on convergence.

```
r1 <- gamlss(R ~ tr(~Fl+A+H+B+loc), data=rent, family=GA, gd.tol=100)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 27824.91
## GAMLSS-RS iteration 2: Global Deviance = 27833.9
## GAMLSS-RS iteration 3: Global Deviance = 27833.9
```

```
r2 <- gamlss(R ~ tr(~Fl+A+H+B+loc), sigma.fo=~tr(~Fl+A+H+B+loc), data=rent,
             family=GA, gd.tol=100, c.crit=0.1)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 27779.77
## GAMLSS-RS iteration 2: Global Deviance = 27754.62
## GAMLSS-RS iteration 3: Global Deviance = 27754.53
```



R code on page 242

Figure 9.22: Visual representation of the neural network model fitted for μ and σ in model `mr4`.

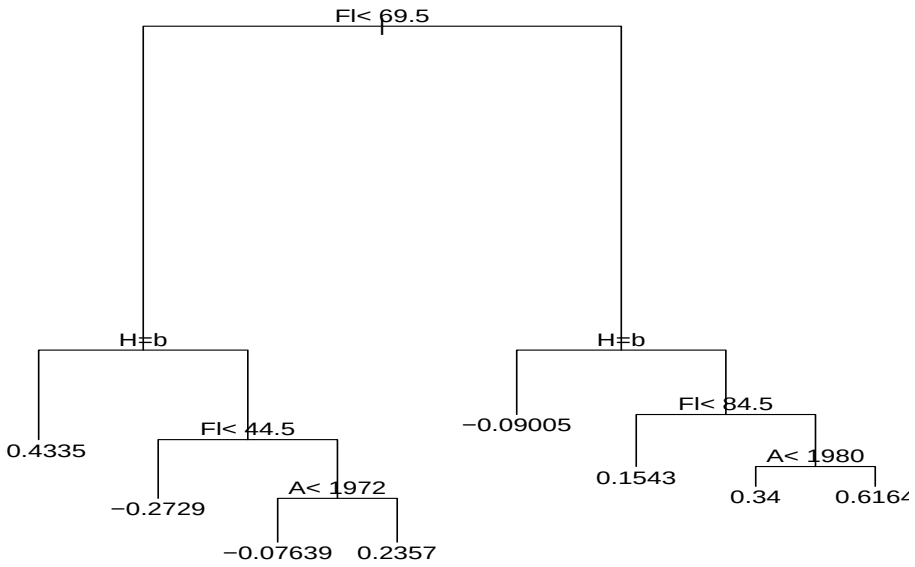
```
AIC(r1,r2)
```

```
##    df      AIC
## r2 12 27778.53
## r1  9 27851.90
```

Note that for the second model we have increased the convergence criterion to 0.1. The plotting of the fitted decision trees can be achieves or by using the `term.plot()` function or by individually plotting the fitted objects as it is demonstrated below:

Figure 9.23

```
term.plot(r2, parameter="mu", pages=1)
```



R code on page 244

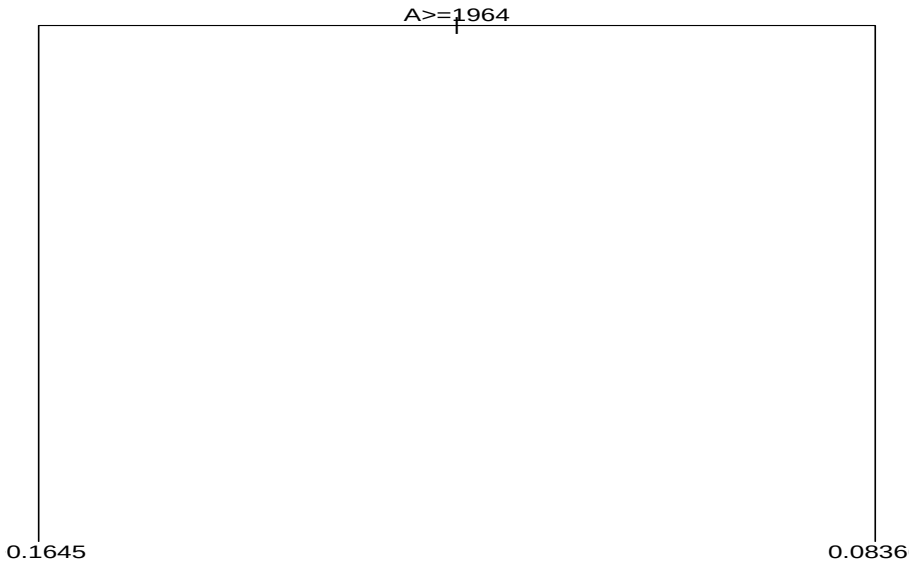
Figure 9.23: Visual representation for the μ parameters of the decision tree model `r2`.

Figure 9.24

```
plot(getSmo(r2, parameter="sigma"))
text(getSmo(r2, parameter="sigma"))
```

9.6.3 Interfacing with `loess()`, the `lo()` function

Under Construction



R code on page 244

Figure 9.24: Visual representation for the μ parameters of the decision tree model `r2`.

9.7 How to add new smooth functions in `gamlss()`

Under Construction

Additive terms	R function names
cubic splines based	`cs()`, `scs()`
decision trees	`tr()`
fractional and power polynomials	`fp()`, `pp()`
free knot smoothing (break points)	`fk()`
`loess`	`lo()`
neural networks	`nn()`
non-linear fit	`nl()`
penalized Beta splines based	`pb()`, `ps()`, `cy()`, pvc()
random effects	`random()`
ridge regression	`ri()`, `ridge()`
Simon Wood's gam	`ga()`

Table 9.2: Additive terms implemented within the **gamlss** packages

Exercises

1. Demonstrate that the penalised least squares quantity in 9.2 is equal to the following augmented least square quantity.

$$\left|\left|\begin{bmatrix}\sqrt{\mathbf{W}} & \mathbf{0}\\ \mathbf{0} & \mathbf{I}\end{bmatrix}\left(\begin{bmatrix}\mathbf{y}\\ \mathbf{0}\end{bmatrix}-\begin{bmatrix}\mathbf{Z}\\ \sqrt{\lambda}\mathbf{D}\end{bmatrix}\boldsymbol{\gamma}\right)\right|\right|^2$$

   where $||\mathbf{z}||^2=\mathbf{z}^\top\mathbf{z}$, $\mathbf{D}$ is any square root of the matrix $\mathbf{G}$ such that $\mathbf{G}=\mathbf{D}^\top\mathbf{D}$ and $\mathbf{W}$ a diagonal matrix of weights. Discuss the implications of this result.

2. Let

$$\widetilde{\mathbf{Z}}=\begin{bmatrix}\mathbf{Z}\\ \sqrt{\lambda}\mathbf{D}\end{bmatrix}$$

   an augmented model matrix as above. Show that the sum of the first n elements on the diagonal of $\widetilde{\mathbf{Z}}\left(\widetilde{\mathbf{Z}}^\top\widetilde{\mathbf{Z}}\right)^{-1}\widetilde{\mathbf{Z}}^\top$ is tr$\left\{\mathbf{Z}\left(\mathbf{Z}^\top\mathbf{Z}+\lambda\mathbf{G}\right)^{-1}\mathbf{Z}^\top\right\}$.

Chapter 10

Random effects

This chapter explains how random effects models can be used within GAMLSS. In particular:

- it introduces the different ways random effects can be fitted within a gamlss models
- it explains the advandages and disadvandages for such modelling
- it uses examples to demostrate the differences

10.1 Introduction

THIS CHAPTER IS UNDER CONSRUCTION .....

There are three distinct ways in which random effects can be introduced within a GAMLSS model:

- random effects at the observational level for μ
- random effects at the factor level for μ and
- random effects for all parameters of the distrbution.

10.2 Random effects models for μ at the observational level

The section describes the use of random effects at the observational level, that is, when there are as many random effects as the number of observations in the data. The main application of random effects of this type is to deal with overdispersion, that is, when extra variability is present in the data which can not be explained purely by the use of the distribution of the response variable itself. We distinguish three types of random effect model at the observational level:

(i) when an an explicit continuous mixture distribution exists.

(ii) when a continuous mixture is not explicit but approximated using Gaussian quadrature points

(iii) when a 'non-parametric' mixture [effectively an finite mixture] is assumed

These different types are described in Sections 10.2.1, 10.2.2 and 10.2.3 respectively.

Section **??** describes the use of random effects at a factor level, that is when we have repeated observations within the same subject and we want the model to take this into account. Both the current and the next Chapters are dealing mainly with random effects affecting the predictor for the parameter μ of a gamlss family distribution. The more general case where a random effect could be present in any of the distributional parameters is discussed in Sections **??** and **??** and in Chapter **??**.

Assume that, given the random effect variable γ, y has conditional probability (density) function $f(\mathbf{Y}|\gamma)$ and marginally γ has probability (density) function $f(\gamma)$. Then the marginal density of Y is given by

$$f(y) = \int f(y|\gamma) f(\gamma) d\gamma. \tag{10.1}$$

Note that γ may be a univariate or multivariate random effect variable. Assume observations $(Y_1, Y_2, \ldots, Y_n)$ are conditionally independent given the observational level random effect variables $(\gamma_1, \gamma_2, \ldots, \gamma_n)$ and that the random effect variables are a random sample from $f(\gamma)$ and therefore also independent, then marginally $(Y_1, Y_2, \ldots, Y_n)$ are independent since

$$\begin{aligned} f_Y(\mathbf{y}) &= \int f(\mathbf{y}|\boldsymbol{\gamma}) f(\boldsymbol{\gamma}) d\boldsymbol{\gamma} \\ &= \int \left[\prod_{i=1}^{n} f(y_i|\gamma_i)\right] \left[\prod_{i=1}^{n} f(\gamma_i)\right] d\boldsymbol{\gamma} \\ &= \int \prod_{i=1}^{n} \left[f(y_i|\gamma_i) f(\gamma_i)\right] d\gamma_1, d\gamma_2, \ldots, d\gamma_n \\ &= \prod_{i=1}^{n} \left[\int f(y_i|\gamma_i) f(\gamma_i) d\gamma_i\right] \\ &= \prod_{i=1}^{n} f_{Y_i}(y_i) \end{aligned} \tag{10.2}$$

where $\mathbf{y}^T = (y_1, y_2, \ldots, y_n)$ and $\boldsymbol{\gamma}^T = (\gamma_1, \gamma_2, \ldots, \gamma_n)$. Hence the likelihood function is a product of the (marginal) likelihoods of each observation Y_i for $i = 1, 2, \ldots, n$, which are obtained by integrating out the random effect for each observation.

10.2.1 Fitting an explicit continuous mixture distributions

For specific combinations of the conditional density function $f(y|\gamma)$ and $f(\gamma)$ the marginal density $f_Y(y)$ can be obtained explicitly and the likelihood maximized directly.

Example 1: Continuous Poisson mixture distribution. Let the conditional distribution of Y given the random effect γ be $PO(\gamma\mu)$ and $\gamma \sim GA(1, \sigma^{1/2})$ then $Y \sim NBI(\mu, \sigma)$. Note that provided γ has mean 1, then μ will be the marginal mean of Y, a desirable property for interpretation of the fitted model.

Example 2: Continuous normal mixture distribution. Let $Y|\gamma \sim N(\mu, \sigma^2\gamma)$ and $\gamma^{-1} \sim \chi^2_\nu$ then $Y \sim TF(\mu, \sigma, \nu)$.

10.2.2 Fitting non-explicit continuous mixture distributions using Gaussian quadrature

For many combinations of $f(y|\gamma)$ and $f(\gamma)$, the marginal density $f_Y(y)$ cannot be obtained explicitly. In this case $f_Y(y)$ can be obtained by numerical integration (at considerable computational cost) or can be obtained approximately by other methods. One approximate method is called Gaussian quadrature, where "Gaussian" refers to the originator of the method and not to any specific (e.g. normal) distribution for $f(\gamma)$. Effectively Gaussian quadrature replace the continuous distribution $f(\gamma)$ with a discrete distribution taking values γ_k with probability π_k for $k = 1, 2, \ldots, K$.

Fitting a normal random effect model in the predictor for μ using Gaussian quadrature

Here we assume that the random effects $\gamma_1, \gamma_2, \ldots, \gamma_n$ (at the observational level) are a random sample from a normal distribution. For $i = 1, 2, \ldots, n$, let $Y_i \sim D(\mu_i, \sigma_i, \nu_i, \tau_i)$ be conditionally independent given random effects γ_i for $i = 1, 2, \ldots, n$ where

$$\begin{aligned} g_1(\boldsymbol{\mu}) &= \boldsymbol{\eta}_1 = \mathbf{X}_1\boldsymbol{\beta}_1 + \boldsymbol{\gamma} \\ g_2(\boldsymbol{\sigma}) &= \boldsymbol{\eta}_2 = \mathbf{X}_2\boldsymbol{\beta}_2 \\ g_3(\boldsymbol{\nu}) &= \boldsymbol{\eta}_3 = \mathbf{X}_3\boldsymbol{\beta}_3 \\ g_4(\boldsymbol{\tau}) &= \boldsymbol{\eta}_4 = \mathbf{X}_4\boldsymbol{\beta}_4. \end{aligned} \tag{10.3}$$

where $\boldsymbol{\gamma}^T = (\gamma_1, \gamma_2, \ldots, \gamma_n)$ and $\gamma \sim N(0, \sigma^2_\gamma)$ independently for $i = 1, 2, \ldots, n$. Let $\gamma_i = \sigma_\gamma Z_i$ then $Z_i \sim N(0, 1)$ independently for $i = 1, 2, \ldots, n$.

Gaussian quadrature effectively approximates the continuous $N(0, 1)$ distribution for each Z_i by a discrete distribution, i.e.

$$Z_i = z_k \qquad \text{with probability} \quad \pi_k \quad \text{for} \quad k = 1, 2, \ldots, K. \tag{10.4}$$

See, for example, Figure 10.2.2 for a visual explanation of the Gaussian quadrature discrete distribution approximation to $N(0, 1)$ when K=10. The model (10.3) with (10.4) can now be considered as a finite mixture of K components in which the prior (or mixing) probabilities π_k's are fixed and known and the z_k's are also fixed and known (once K the total number of quadrature points has been chosen). Hence $g_1(\mu_{ik}) = \mathbf{x}^T_{ik}\boldsymbol{\beta}_1 + z_k\sigma_k\boldsymbol{\gamma}$ with probability π_k with $k = 1, 2, \ldots, K$.

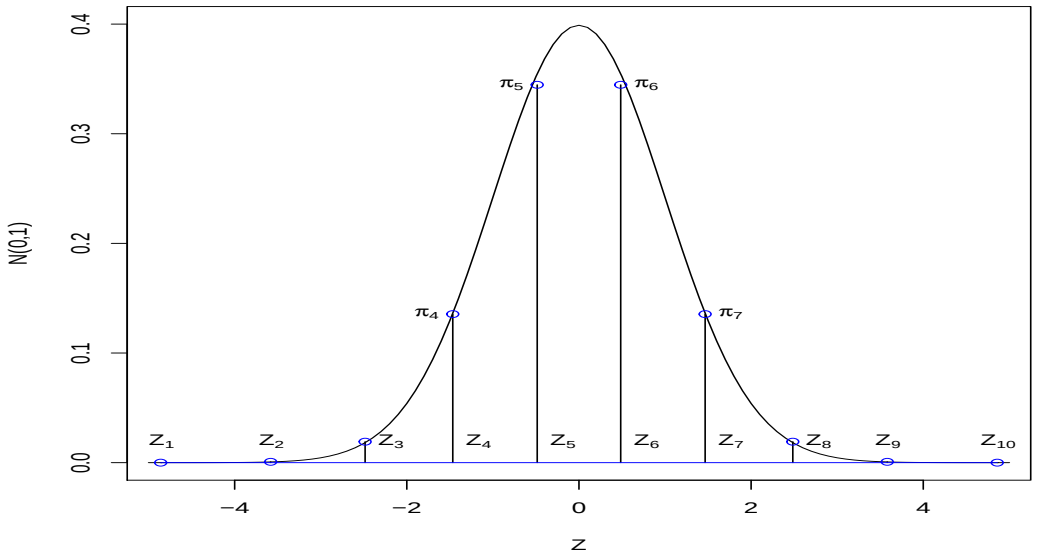


Figure 10.1: Plot showing an example of non-parametric (discrete) distribution.

10.2.3 Non parametric random effects models

Non parametric random intercept in the predictor for μ

Here assume that the random effects $\gamma_1, \gamma_2, \ldots, \gamma_n$ in model (10.3) are a random sample from a (non-parametric) distribution which is modelled as a discrete distribution given by

$$\gamma_i = u_k \qquad \text{with probability} \quad \pi_k \quad \text{for} \quad k = 1, 2, \ldots, K \tag{10.5}$$

for $i = 1, 2, \ldots, n$. The u_k's and π_k's are assumed to be fixed unknown constants. See Figure 10.2.3 for an example plot of a 'non-parametric' distribution (10.5) with $K = 5$. The resulting (marginal) model for y_i is just a finite mixture of GAMLSS models with parameters $(\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \boldsymbol{\beta}_3, \boldsymbol{\beta}_4)$ in common, but different intercept parameters $(u_1, u_2, \ldots, u_K)$ in the predictor for μ. The model can be fitted using the EM algorithm of Section 7.5. and using the R function `gamlssNP()`.

Non parametric random coefficients in the predictor for μ

Model (10.3) can be amended to a model with non-parametric random intercept and slope in the predictor for μ, i.e.

$$g_1(\boldsymbol{\mu_i}) \quad = \quad \boldsymbol{\eta}_1 = \mathbf{x}_{1i}^T \boldsymbol{\beta}_1 + \boldsymbol{\gamma}_0 + \boldsymbol{\gamma}_1 \mathrm{x}_{1i} \tag{10.6}$$

(10.7)

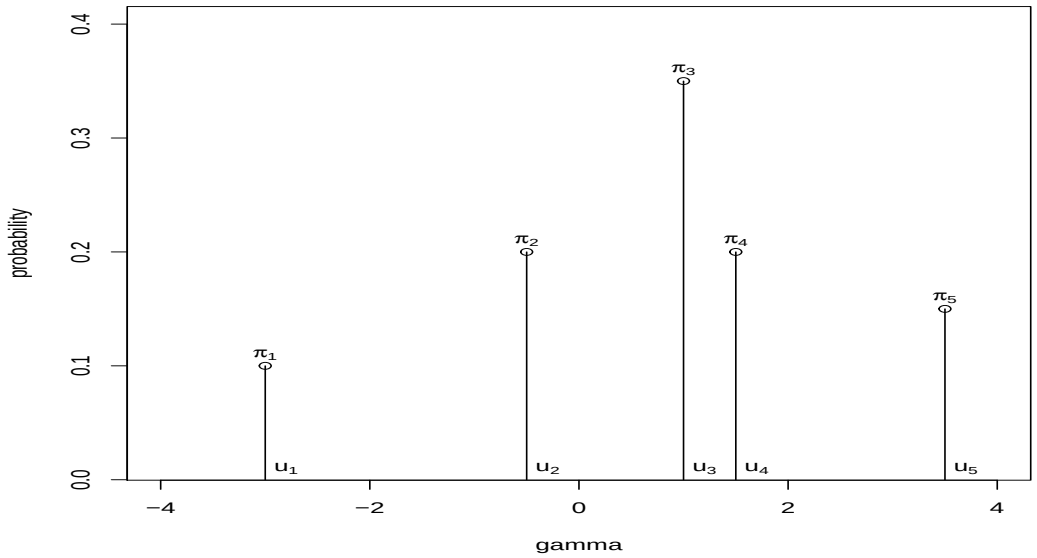


Figure 10.2: Plot showing how the continuous distribution NO$(0, 1)$ is approximated by Gaussian quadrature with $K = 10$

for $i = 1, 2, \ldots, n$ where $\boldsymbol{\gamma}_0^T = (\gamma_{01}, \gamma_{02}, \ldots, \gamma_{0n})$, $\boldsymbol{\gamma}_1^T = (\gamma_{11}, \gamma_{12}, \ldots, \gamma_{1n})$ and $(\gamma_{0i}, \gamma_{1i})$ for $i = 1, 2, \ldots, n$ are a random sample from a bivariate 'non-parametric' distribution, taking values (u_{0k}, u_{1k}) with probability π_k for $k = 1, 2, \ldots, K$, i.e. $(\gamma_{0i}, \gamma_{1i}) = (u_{0k}, u_{1k})$, with probability π_k for $k = 1, 2, \ldots, K$ for $i = 1, 2, \ldots, n$. As an example of a two dimensional non-parametric distribution see Figure (10.2.3) where a hypothetical 'non-parametric' distribution is plotted with $K = 10$. The prior probabilities π_k, for $k = 1, 2, \ldots, K$. are assumed constant over i. Again the resulting model is a finite mixture model (with parameters in common) and can be fitted using the EM algorithm of Section 7.5. Additional random coefficients can easily be included in the predictor for μ.

The function `gamlssNP()` can be used to fit the model using the argument `random` to declare which of the explanatory variables for μ should have random coefficients.

10.2.4 Non parametric random coefficients in the predictor for all distribution parameters

Model (10.3) can be amended to include 'non-parametric' random coefficients (e.g. intercept and slopes) in the predictor for any one or more of the distributional parameters μ, σ, ν and τ of a GAMLSS model. The model is fitted using the EM algorithm of Section 7.5. For example model (10.3) could be amended to model

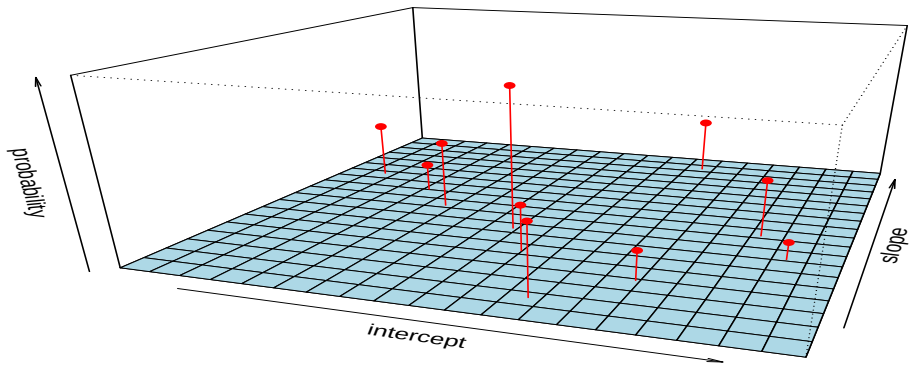


Figure 10.3: Plot showing a non parametric mixture distribution in two dimensions with $K = 10$

$$\begin{aligned} g_1(\mu_i) &= \mathbf{x}_{1i}^T\boldsymbol{\beta}_1 + \gamma_{10} + \gamma_{11}\mathtt{x}_{1i} \\ g_2(\sigma_i) &= \mathbf{x}_{2i}^T\boldsymbol{\beta}_2 + \gamma_{20} \\ g_3(\nu_i) &= \mathbf{x}_{3i}^T\boldsymbol{\beta}_3 \\ g_4(\tau_i) &= \mathbf{x}_{4i}^T\boldsymbol{\beta}_4. \end{aligned}$$

for $i = 1, 2, \ldots, n$. In the function `gamlssNP()` the above model for σ can be fitted by adding the factor `MASS` in the formula for the predictor of σ, e.g. `sigma.fo=~x+MASS`.

10.3 Random effects models for μ at the factor level

Part V

Model selection and diagnostics

Chapter 11

Model selection techniques

This chapter explains the model selection techniques in `gamlss`. In particularly it explains:

1. The different components of a GAMLSS model important for selecting an appropriate model
2. The different stepwise selection functions and techniques used for selecting explanatory terms
3. The different techniques for selecting smoothing parameters

This chapter is important to understand the statistical modelling process within GAMLSS.

11.1 Introduction: Statistical model selection

This chapter will discuss what available statistical modelling techniques and functions exist within the GAMLSS framework. We start with a general discussion about selecting an appropriate statistical model and then we move specifically to GAMLSS models.

Statistical models are built to:

- explore the data where no theory exists, *exploratory* models,
- explain or verify a theory, *explanatory* models,
- predict future values, *predictive* models

or any combination of the above situations. In general it is recognised that an *overfitted* model, that is, a fitted model which is very close to the current data, is not very good for prediction. Therefore in any model selection and depending on the purpose of the study a balance has to be made between over fitting (over-interpreting the current data) and underfitting (under-interpreting the data). In statistical inference terms, this turns out to be a balance between *variance* and *bias* of the estimators. Overfitted model estimators have big variance and therefore are bad for predicting future values while underfitted model estimators are biased but with smaller variance and can therefore sometimes be better for prediction.

Let $\mathcal{M}$ be a statistical model. For a parametric statistical model and within a likelihood based inferential procedure, each fitted model $\mathcal{M}$ can be assessed by its fitted global deviance, GD, given by $GD = -2\ell(\hat{\boldsymbol{\theta}})$ where $\boldsymbol{\theta}$ are the parameters of the model, and $\ell()$ is the fitted (or maximized) log-likelihood function.

Let $\mathcal{M}_0$ and $\mathcal{M}_1$ be two different statistical models with fitted global deviances GD_0 and GD_1 and degrees of freedom df_0 and df_1 respectively.

Definition: Model $\mathcal{M}_0$ is *nested* within $\mathcal{M}_1$ if $\mathcal{M}_0$ is a subclass of model $\mathcal{M}_1$.

Two nested parametric statistical models, $\mathcal{M}_0$ and $\mathcal{M}_1$, may be compared using the (generalized likelihood ratio) test statistic

$$\Lambda = GD_0 - GD_1$$

which has an asymptotic χ^2-squared distribution under the null hypothesis that the correct model is $\mathcal{M}_0$, with degrees of freedom $d = df_0 - df_1$. (given that some regularity conditions reference, like that the maximum does not occur on the boundary space of the parameters, are satisfied).

When the statistical models $\mathcal{M}_0$ and $\mathcal{M}_1$ contain *non-parametric* additive terms the same test can be used as a guide to fitted model selection in the same way that Hastie and Tibshirani [1990] (Ch 3.9) compare 'nested' Generalized Additive Models (GAM) fits. The degrees of freedom used here is the trace of the smoothing matrix $\mathbf{S}$ in the fitting algorithm, called the *effective* degrees of freedom, see Chapter **??** or Hastie and Tibshirani [1990].

For comparing non-nested GAMLSS models, to penalize over-fitting the generalized Akaike Information Criterion (GAIC), Akaike [1983], can be used. This is obtained by adding to the fitted deviance a fixed penalty k for each effective degree of freedom used in a model, i.e. GAIC(k) = GD + ($k \times df$), where df denotes the total effective degrees of freedom used in the model and GD is the fitted global deviance. The model with the smallest value of the criterion GAIC(k) is then selected. The Akaike information criterion (AIC), Akaike [1974], and the Schwartz Bayesian criterion (SBC), Schwarz [1978], are special cases of the GAIC(k) criterion corresponding to $k = 2$ and $k = \log(n)$ respectively. The two criteria, AIC and SBC, are asymptotically justified as predicting the degree of fit in a new data set, i.e. approximations to the average predictive error. Justification for the use of SBC comes also as a crude approximation to Bayes factors, Raftery [1996, 1999]. In practice it is usually found that the original AIC leads to overfitting in model selection while the SBC leads to underfitting. Our experience suggests that a value of the penalty k in the range $2.5 \leq k \leq 4$ works well. **?** suggested using $k \approx 2.8$. A selection of different values of k, e.g. $k = 2, 2.5, 3, 3.5, 4$, could be used in turn to investigate the sensitivity or robustness of the model selection to the choice of the value of the penalty k. Using GAIC(k) allows different penalties k to be tried for different modelling purposes. The sensitivity of the selected model to the choice of k can also be investigated. Claeskens and Hjort [2003] consider a focused information criterion (FIC) in which the criterion for model selection depends on the objective of the study, in particular on the specific parameter of interest.

Cross validation techniques play an important role in model selection especially when prediction is important. The idea of K-fold cross validation is simple. First randomly divide your data into K subsets, $S_1, S_2, \ldots, S_K$.

- Then, for $j = 1, 2, \ldots, K$, omit the S_j data set and fit a model to all other data.
- Calculate a measure of goodness of predictive fit to the S_j data set.

- Combine the K measures of goodness of fit.
- Choose between models using the combined measurement of goodness of fit.

The comparison can be done using the deviance or any other measure of disparity and the model with the smallest overall value is the best for prediction purposes (for the chosen measure of discrepancy). A cross validation performed this way is called a *K-fold cross validation*. A *simple cross validation* is defined when only one observation is omitted in each fit and hence $k = n$, the number of observations. For linear models the simple cross validation can be achieved efficiently without having to refit n models, since the 'leave one out' fitted values can be calculated easily from the fitted values and the diagonal of the hat matrix of the original fit including all observations.

The GAIC and cross validation techniques are used for reasonably small data sets where the full data sample is used for both model fitting (minimizing GD) and for model selection (minimizing a penalized criterion, e.g. AIC or SBC, or a cross valuation criterion). For very large data sets, the data could be randomly split into:

(i) training data set

(ii) validation data set

(iii) test data set

The training data is used used for model fitting, the validation data set is used for model selection and the test data set is used for model assessment. This split is now routinely available in data mining statistical packages such as SAS Enterprise Miner SAS Institute Inc. [2000]. Some of these procedure are now implemented in the **gamlss** packages and they will described later in this chapter.

Inference about quantities of interest can be made either conditionally on a single selected 'final' model or by averaging between selected models. If the purpose of the study is to describe the data parsimoniously, then a single 'final' model is usually sufficient. Conditioning on a single final model was criticized by Draper [1995] and Madigan and Raftery [1994] since it ignores model uncertainty and generally leads to the underestimation of the uncertainty about quantities of interest. Averaging between selected models can reduce this underestimation, Hjort and Claeskens [2003].

Different model selection strategies can be used to build a statistical model but more importantly the determination of the *model adequacy should always be carried out with respect to the substantive questions of interest* and not in isolation. This means that different problems could possibly require different model strategies.

11.2 GAMLSS model selection

Let $\mathcal{M} = \{\mathcal{D}, \mathcal{G}, \mathcal{T}, \Lambda\}$ represent a GAMLSS model as defined in section **??**. The components of $\mathcal{M}$ are defined as follows:

(i) $\mathcal{D}$ specifies the distribution of the response variable,

(ii) $\mathcal{G}$ specifies the set of link functions,

(iii) $\mathcal{T}$ specifies the terms appearing in all the predictors for μ, σ, ν and τ,

(iv) Λ specifies the smoothing hyper-parameters which determine the amount of smoothing in the $h_{jk}()$ functions of equation (**??**) .

In the search for an appropriate GAMLSS model for any new data set, all the above four components have to be specified as objectively as possible.

We will next discuss how the components $\mathcal{D}$, Λ, $\mathcal{T}$ and Λ can be specified analysing different data.

11.2.1 Component $\mathcal{D}$: Selection of the distribution

The selection of the appropriate distribution can be done in two stages, the *fitting* stage and the *diagnostic* stage. The fitting stage involves the comparison of different fitted models using a generalised Akaike information criterion, (GAIC). The model with the smallest value of the criterion GAIC(k), for a chosen value of k (see section 11.1) is then selected.

The diagnostics stage involves the use of *worm plots*. Worm plots were introduced by van Buuren and Fredriks [2001] and are in effect de-trended normal QQ plots of the quantile residuals (i.e. z-scores) see section **??** for more details. The worm plot allows detection of inadequacies in the model globally or within specific ranges of one (or two) explanatory variable.

11.2.2 Component $\mathcal{G}$: Selection of the link functions

The selection of the link function for each distribution parameter is usually determined by the range of the parameter in hand. For example in the Pareto II, (`PARETO2`), distribution both μ and σ take values in the positive line so a log link function is a natural way of ensuring that parameters μ and σ remain positive (whenever the values of their predictors). For a normal distribution, `NO`, $-\infty < \mu < \infty$ and $0 < \sigma < \infty$, so an identity for μ and a log link for σ insures that μ and σ are always within their ranges.

There are occasions in which the choice of the link function is important from the interpretation point of view. For example if we believe that the explanatory variables effect the distribution parameter multiplicatively rather than additively, then a log link is more appropriate.

The choice of link may improve the model fit considerably. Different link functions can be compared directly using the global deviance. The best link function results in the lowest deviance.

11.2.3 Component $\mathcal{T}$: Selection of the additive terms in the model

Let $\mathcal{X}_i$ be a pool of terms available for consideration for the parameter $\boldsymbol{\theta}_i$ for $i = 1, 2, 3, 4$ where $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3, \boldsymbol{\theta}_4) = (\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\nu}, \boldsymbol{\tau})$. Typically $\mathcal{X}_i$ will contain both linear and smoothing additive terms. For example, let f_1 and f_2 represent factors and x_1, x_2, x_3 and x_4 continuous explanatory variables. Then, for example,

$$\mathcal{X}_i = \{f_1 * f_2 + s(x_1) + s(x_2, x_3) + x_4\}$$

allows second order interactions for the two factors, a smooth functions for x_1, a smooth interaction for x_2, x_3 and linear term for x_4. There are a few points to emphasise here:

- For a given distribution for the response variable, the selection of the terms has to be done for **all** the parameters of the assumed distributions, not only the location parameter. The usual *forward*, *backward* and *stepwise* procedures can be applied here for each parameter but also some thought has to be given on how those procedures can be applied when we choose terms for all the parameters.
- The additive terms can influence the distribution parameter in different ways. For example in above example the interaction of the factors f_1 and f_2 affects the parameter of interest. The variable x_4 influence the parameter linearly, the variable x_1 non-linearly while a smooth non-linear interaction between x_2 and x_3 affects the parameter of interest.
- The size of available terms $\mathcal{X}_i$, relatively to the number of observations in the sample matters as far as selection of terms is concerned. For example, if the number of continuous explanatory variables is small, say 5, all $2^5 = 25$ different combinations of how those 5 variables can influence a parameter can be tried. On the other hand when we are dealing with say 50 continuous explanatory variables, there are $2^{50} = 1.13 \times 10^{15}$ different combinations which can not all be fitted, so we have to implement a different strategy.

There are several functions within GAMLSS to assist with selecting explanatory variable terms when all the data points are used for the selection of variables (see Section 11.2.5 for when this is not the case). The basic functions are `addterm()` and `dropterm()`, which allow the addition or removal of one term in a predictor of a parameter model respectively. The functions `add1()` and `drop1()` are identical to `addterm()` and `dropterm()` respectively, but used different default values for one of the arguments, see section 11.3. The functions `addterm()` and `dropterm()` are the building blocks for the function `stepGAIC()` suitable for stepwise selection of terms for one of the distribution parameters of a GAMLSS models using a Generalized Akaike Information Criterion (GAIC).

There are many different strategies that could be applied for the selection of the terms used to model **all** the parameters μ, σ, ν and τ of a GAMLSS model. In the current implementation we have two strategies for selecting a terms for all the parameters. We call them strategy A and strategy B. They are implemented in the `stepGAICAll.A()` and `stepGAICAll.B()` function respectively. BOOSTING, and shrinkage methods

11.2.4 Component Λ: Selection of the smoothing parameters

Each smoothing term selected for any of the parameters of the distribution has at least one smoothing (or hyper) parameter λ associated with it. We denote by Λ the set of all smoothing parameters e.g. $\Lambda = \{\lambda_{\mu,1}, \lambda_{\mu,2}, \lambda_{\sigma,1}, \lambda_{\nu,1}\}$,

The smoothing parameters can be fixed or estimated from the data. The standard way of fixing the smoothing parameters, as suggested in Hastie and Tibshirani [1990], is by fixing the effective degrees of freedom for smoothing. A lot of the smoothing procedures within the **gamlss** packages allow the user to do that.

More generally it is desirable to estimate the smoothing parameter automatically. The following are three common methods of estimating the smoothing parameters:

- Generalised cross validation (GCV),
- GAIC ,

- Maximum likelihood method.

Each method can be done in two ways:

locally: when the method is applied each time within the iterative GAMLSS algorithm.

globally: when the method is applied outside of the iterative GAMLSS algorithm

Table 11.1 shows where information about the different methods can be obtained.

Global	Method	Reference
Global	ML /REML (e.g. Laplace)	Rigby and Stasinopoulos [2005]
Global	GAIC (e.g. AIC, SBC)	Rigby and Stasinopoulos [2004, 2006a]
Global	Validation Global Deviance (VGD)	Stasinopoulos and Rigby [2007]
Local	ML	Rigby and Stasinopoulos [2013]
Local	GAIC	Rigby and Stasinopoulos [2013]
Local	Generalized Cross Validation (GCV)	Wood [2006]

Table 11.1: Showing references for the different approaching of choosing the smoothing parameters

In our experience the local methods are much faster and often produce similar results to the global methods. The global methods can sometimes be more reliable.

11.2.5 Selection of all components using a validation data set

For large data sets, within the GAMLSS framework, the statistical modeller can afford to split the data into different parts. For example:

i) the training data could be used for model fitting (minimizing its GD)

ii) the validation data could be used for model selection, in particular selection of the distribution, link functions, predictor terms and smoothing parameters (by minimizing its GD, denoted by VGD)

iii) the test data could be used for the assessment of the predictive power of the model chosen by (ii) and fitted by (i) and applied to the test data (again using its GD, denoted by TGD).

There are several functions within the **gamlss** package to assist the model selection in those cases. For example the function `gamlssVGD()` fits a model to the training data set and then calculates the validation global deviance for the validation data set. Different models fitted this way can be compared using the function `VGD()` which is behaving similarly to `GAIC()` function. If we already have GAMLSS fitted models, and we want to see how well they are doing on a new (validation or test) data set, then the function `getTGD()` can be used to get their validation or test global deviance and the function `TGD()` can be used to compared them.

Component	All data	K-fold cross validation	Validation and test data
$\mathcal{D}$	GAIC()* wp() *	gamlssCV(), CV()	gamlsVGD(), VGD() getTGV() TGD()
$\mathcal{G}$	deviance() *	gamlssCV(), GV()	as above
$\mathcal{T}$	drop1(), add1(), add1ALL(), drop1ALL(), stepGAIC() stepGAICAll.A() stepGAICAll.B()	gamlssCV(), CV()	drop1TGD() add1TGD() stepTGD()
Λ global	findhyper() optim()*	optim()*	optim()*

Table 11.2: Showing the different model selection functions described in this Chapter according to which part of a GAMLSS model used and according to different data set up. Functions with asterisk are not covered in this Chapter

The basic functions for selection of variables, when we have training and validation data sets exist are `add1TGD()` and `drop1TGD()` which allow the inclusion or exclusion of a single term in the model. The function for stepwise selection of variables for a single parameter of the distribution of the response is called `stepTGD()`. At the moments there are no functions implementing different strategies for selecting explanatory variables for all the parameters of the response.

11.2.6 Summary of the GAMLSS functions for model selection

Table 11.2 provides a summary of the different model selection functions described in this Chapter according to which part of the GAMLSS model can be used, and according to which purpose. The functions with an added asterisk are function covered in other part of the book and therefore are not described specifically in this Chapter.

The next four sections of this Chapter describe the **gamlss** package functions which can be used if all the data points are used for model selection. Section 11.3 describe the `addterm()` and `dropterm()` functions which are the building blocks for a full model selection strategy. The `add1()` and `drop1()` functions are identical to the `addterm()` and `dropterm()` with different default values for the argument `test`.

11.3 The `addterm()` and `dropterm()` functions

The functions `addterm()` and `dropterm()` are generic **R** S3 object functions with their original definitions defined in the package **MASS** of of Vendable and Ripley (2002). This package is

attached, so their method for classes `gamlss` can be used. Note that the functions `addterm()` and `dropterm()` have a *parallel* argument which can be used for parallel computations. This can be beneficial for large data sets when the fitting of each individual model can take several minutes (assuming of course that the computer have multiple CPUs).

The `dropterm()` and `addterm()` functions in GAMLSS have the following arguments

`object` a gamlss object.

`scope` a formula giving terms which might be dropped or added. For the function `dropterm` the default is the model formula. For the function `addterm` the `scope` is a formula specifying a maximal model which should include the current one. Only terms that can be dropped or added while maintaining marginality are actually tried.

`what` the parameter of the distribution (equivalent to `parameter`)

`parameter` a different way to specify the parameter of the distribution rather than `what`

`scale` scale is not used in `gamlss`

`test` it takes value `"none"` for no test and `"Chisq"` for a χ^2 test statistic relative to the original model. Note that the default values is `"none"` for the functions `dropterm()` and `addterm()` while it is `"Chisq"` for the equivalent `drop1()` and `add1()` functions.

`k` the penalty for each extra degree of freedom used in the GAIC. Note `k = 2` gives the original AIC while `k = log(n)` gives SBC.

`sorted` If `TRUE`, (the default), the results are sorted in the order of the GAIC from the lowest (the best model) to the highest (the worst model).

`trace` if 'TRUE', (the default), additional information may be given on the fits as they are tried.

`parallel` The type of parallel operation to be used with alternatives "no", "multicore" and "snow". The default is "no".

`ncpus` the number of processes to be used in parallel operation: typically one would chose this to the number of available CPUs in the computer.

`cl` This is optional name of a parallel or snow cluster if `parallel = "snow"` is used. If the argument is not supplied, a cluster on the local machine is created for the duration of the call.

... arguments passed to or from other methods.

The functions `drop1()` and `add1()` are identically to the functions `dropterm()` and `addterm()` respectively with the argument `test="Chisq"` is used.

In order to demonstrate how `dropterm()` and `addterm()` (or their equivalent `drop1()` and `add1()`) are working we are using the US pollution data set taken from Hand *et al.* (1994) and the `aids` data.

11.3.1 drop1()

Data summary: US pollution data

R data file: usair in package **gamlss** of dimensions 41×7

variables

- y : sulphur dioxide concentration in air in mgs. per cubic meter
- x1 : average annual temperature in degrees F
- x2 : number of manufacturers employing ¿ 20 workers
- x3 : population size in thousands
- x4 : average annual wind speed in miles per hour
- x5 : average annual rainfall in inches
- x6 : average number of days rainfall per year

purpose: to demonstrate term selection techniques

Preliminary analysis has shown that it is better to model the distribution of the response variable Y in the usair data using the gamma rather the normal distribution. We start by fitting the full linear model for μ including all six explanatory variables:

```
data(usair)
mod1<-gamlss(y~., data=usair, family=GA, trace=FALSE)
summary(mod1)
```

```
## ******************************************************************
## Family:  c("GA", "Gamma")
##
## Call:  gamlss(formula = y ~ ., family = GA, data = usair, trace = FALSE)
##
##
## Fitting method: RS()
##
## ------------------------------------------------------------------
## Mu link function:  log
## Mu Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  7.3164944  1.3681744   5.348 6.61e-06 ***
## x1          -0.0622829  0.0168679  -3.692 0.000798 ***
## x2           0.0013416  0.0003506   3.826 0.000550 ***
## x3          -0.0008132  0.0003546  -2.294 0.028323 *
## x4          -0.1562766  0.0557698  -2.802 0.008429 **
## x5           0.0196006  0.0101839   1.925 0.062928 .
## x6           0.0002016  0.0047788   0.042 0.966601
## ---
## 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)  -0.9022     0.1713  -5.268 8.36e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## ------------------------------------------------------------------
## No. of observations in the fit:  41
## Degrees of Freedom for the fit:  8
##       Residual Deg. of Freedom:  33
##                       at cycle:  2
##
## Global Deviance:     303.1602
##             AIC:     319.1602
##             SBC:     332.8687
## ******************************************************************
```

The '.' in the command selects all variables other than y in the `data.frame` `usair` as explanatory variables. Now we use the `drop1()` function to check whether any linear terms can be dropped.

```
dd<-drop1(mod1)
dd
```

```
## Single term deletions for
## mu
##
## Model:
## y ~ x1 + x2 + x3 + x4 + x5 + x6
##        Df    AIC     LRT  Pr(Chi)
## <none>    319.16
## x1      1 327.58 10.4245 0.001244 **
## x2      1 326.92  9.7557 0.001788 **
## x3      1 321.39  4.2299 0.039717 *
## x4      1 324.08  6.9247 0.008501 **
## x5      1 320.57  3.4141 0.064642 .
## x6      1 317.16  0.0017 0.966960
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The above output gives also the likelihood ratio test Λ, (LRT given in Section 11.1) and its Chi-square p-values for removing each of the six variables from the full model. In the specific example above, given all other linear terms in the model, the variable $x6$ is the first to be dropped since it has the highest p-values, 0.967, given by column `Pr(Chi)`, and so is the least significant.

For a full parametric models (like the one above) and with only continuous terms in the model the Chi-square using `drop1()` and the t-values using `summary()` should produce identical con-

clusions. However when factors or smooth terms are in the model, `summary()` do not provide the right information for testing if a term can be excluded or not from the model given the rest of the terms in the model. This is where `drop1()` can very useful in model selection. Next we demonstrate the use of `drop1()` when a smoother and a factor are in the model using the `aids` data sets.

```
data(aids)
aids1<-gamlss(y~qrt+pb(x), data=aids, family=NBI, trace=FALSE)
summary(aids1)
```

```
## ******************************************************************
## Family:  c("NBI", "Negative Binomial type I")
##
## Call:  gamlss(formula = y ~ qrt + pb(x), family = NBI, data = aids,
##     trace = FALSE)
##
## Fitting method: RS()
##
## ------------------------------------------------------------------
## Mu link function:  log
## Mu Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.659646   0.057763  46.044  < 2e-16 ***
## qrt2        -0.162258   0.046546  -3.486  0.00139 **
## qrt3         0.024024   0.045395   0.529  0.60015
## qrt4        -0.121794   0.045385  -2.684  0.01124 *
## pb(x)        0.093858   0.001597  58.786  < 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)   -5.272      0.433  -12.18 7.82e-14 ***
## ---
## 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:  11.58886
##       Residual Deg. of Freedom:  33.41114
##                       at cycle:  5
##
## Global Deviance:     366.9258
```

```
##             AIC:     390.1036
##             SBC:     411.0407
## ********************************************************************
```

```
drop1(aids1)
```

```
## Single term deletions for
## mu
##
## Model:
## y ~ qrt + pb(x)
##            Df    AIC     LRT   Pr(Chi)
## <none>        390.10
## qrt    4.0778 403.95  22.003 0.0002168 ***
## pb(x)  6.5889 576.91 199.983 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The t-tests provided from the `summary()` function should be use with caution. For example the t-values of 0.529 of the third quarter `qrt3` is testing whether the third quarter is significant different from the first quarter [which here is the value given by the `(Intercept)`]. This is not a test whether overall the factor `qrt` contributes significantly to the model or not. The `drop1()` Chi-square test for `qrt` provides this and with a p-value less than 0.001 it shows that `qrt` is highly significant.

A more serious problem could arise for the misinterpretation of the value of the t-statistic provided for smoothing terms. In our case the smoothing term for `pb(x)` in the `summary()` table has a t-value. This is **not** a test whether the overall smooth function for x is significant or not. Instead this test checks whether the linear part in x is significant, given the factor `qrt` and the non-linear contribution of x are already in the model. That is a rather peculiar test and arises due to the way that the GAMLSS backfitting algorithm works. The real question is whether the smoother for x is significant given `qrt` and this is given from the Chi-square test output of `drop1()`. With a p-value of close to 0 the smooth function for x has a highly significant contribution to the model.

11.3.2 `add1()`

To demonstrate the function `add1()` consider adding a two way interaction term into the linear model `mod1` of the `usair` data. Note that when `add1()` is used the `scope` argument has to be defined explicitly.

```
add1(mod1, scope=~(x1+x2+x3+x4+x5+x6)^2)
```

```
## Single term additions for
## mu
##
## Model:
## y ~ x1 + x2 + x3 + x4 + x5 + x6
##        Df    AIC     LRT   Pr(Chi)
## <none>    319.16
```

```
## x1:x2   1 320.09  1.0689 0.3012045
## x1:x3   1 319.40  1.7626 0.1843028
## x1:x4   1 320.60  0.5623 0.4533271
## x1:x5   1 316.94  4.2226 0.0398901 *
## x1:x6   1 320.93  0.2351 0.6277906
## x2:x3   1 320.48  0.6786 0.4100846
## x2:x4   1 319.75  1.4138 0.2344256
## x2:x5   1 318.17  2.9873 0.0839194 .
## x2:x6   1 321.13  0.0310 0.8603147
## x3:x4   1 317.38  3.7783 0.0519200 .
## x3:x5   1 320.19  0.9672 0.3253680
## x3:x6   1 320.85  0.3061 0.5800599
## x4:x5   1 307.07 14.0870 0.0001745 ***
## x4:x6   1 320.33  0.8346 0.3609322
## x5:x6   1 318.74  2.4188 0.1198894
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Among the two way interactions `x4:x5` is highly significant with a p-value of less that 0.001.

The next code is to demonstrate how smoothers can be used. In the beginning we create a formula containing all the explanatory variables in the data `usair` using the smoother `pb()`. Them we fit the null model containing only the constant and then we add each smoother one at a time. The resulting LRT shows whether the smooth functions of the explanatory variables can explain on their own the response variables. With p-values less than 0,05 shows that all the explanatory variables can explain the response well.

```
FORM <- as.formula( paste("~",paste(paste(paste("pb(", names(usair[-1]),
                 sep=""),")", sep=""), collapse="+")))
FORM
```

```
## ~pb(x1) + pb(x2) + pb(x3) + pb(x4) + pb(x5) + pb(x6)
```

```
mod0 <- gamlss(y~1, data=usair,family=GA, trace=FALSE )
add1(mod0, scope=FORM)
```

```
## Single term additions for
## mu
##
## Model:
## y ~ 1
##            Df    AIC     LRT   Pr(Chi)
## <none>        353.71
## pb(x1) 1.0000 338.04 17.6792 2.615e-05 ***
## pb(x2) 1.0000 343.05 12.6660 0.0003724 ***
## pb(x3) 1.6664 348.02  9.0249 0.0073307 **
## pb(x4) 3.3633 345.59 14.8494 0.0028119 **
## pb(x5) 3.0057 341.75 17.9793 0.0004471 ***
## pb(x6) 1.1863 343.82 12.2658 0.0006515 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

11.4 The stepGAIC() function

In order to build a model for any of the parameters of the distribution of the response variable using a forward, backward or stepwise procedure the functions `stepGAIC()` can be used. `stepGAIC()` is based on the function `stepAIC()` given in the library MASS of Venables and Ripley [2002] (where more details and examples of the function can be found). The additional argument `parameter` is designed to allow selection of terms for a specific parameter of the distribution. The function has been also changed to allow parallel computations. [The older version of `stepGAIC()` function with no parallel facilities can be found under the name `stepGAIC.VR()`]. The main arguments of the function `stepGAIC()` are:

`object` a `gamlss` object which is used as the initial model in the stepwise search

`scope` The scope defines the range of models examined in the stepwise search. The set of models searched by `stepGAIC()` is determined by the `scope` argument and its `lower` and `upper` components. The scope should be either a single formula, or a list containing components `upper` and`lower`, both formulae. The terms defined by the formula in `lower` component are always included in the model. The formula in `upper` is the most complicated model that the procedure would consider. The lower model must be a sub-model of the upper model. The initial fitted model specified in the `object` option must be the lower or upper model or a model lying between them. If the `scope` is missing then a backward elimination starts from the model define by the `gamlss` object.

`direction` the mode of stepwise search, can be one of `both`, `backward`, or `forward`, with a default of `both` which performs a stepwise model selection. If the `scope` argument is missing the default for `direction` is `backward`

`trace` if positive, information is printed during the running of `stepGAIC`. Larger values may give more information on the fitting process

`keep` a filter function whose input is a fitted model object and the associated `AIC` statistic, and whose output is arbitrary. Typically `keep` will select a subset of the components of the object and return them. The default is not to keep anything.

`steps` the maximum number of steps to be considered. The default is 1000 (essentially as many as required). It is typically used to stop the process early.

The extra arguments `what`, `parameter`, `k`, `parallel`, `ncpus` and `cl` operate similarly to the ones described in Section 11.3 for `dropterm()` and `addterm()` functions.

The set of models searched by `stepGAIC()` is determined by the `scope` argument and its `lower` and `upper` components. The `lower` and `upper` components are model formulae. The terms defined by the formula in the `lower` component are always included in the model. The formula in `upper` is the most complicated model that the procedure would consider. The lower model must be a sub-model of the upper model. The model given the the argument `object` must

be the lower or upper model or a model between them. That is, the fitted model specified should lie between the `lower` and `upper` models. If the `scope` is missing then a backward elimination starts from the model define by the `gamlss` object. In the following example a backward elimination is performed on the model given by `mod1`. Note that `mod2` has a new component called `anova` showing the steps taken in the search of the model.

11.4.1 Selecting model for μ

In the following example a backward elimination is performed on the model given by `mod1`. Note that `mod2` has a new component called `anova` showing the steps taken in the search of the model.

```
mod2<-stepGAIC(mod1)
mod2$anova
```

```
## Distribution parameter:  mu
## Start:  AIC= 319.16
##  y ~ x1 + x2 + x3 + x4 + x5 + x6
##
## . . .
## Step:  AIC= 317.16
##  y ~ x1 + x2 + x3 + x4 + x5
##
##        Df    AIC
## <none>    317.16
## - x3    1 319.39
## - x4    1 322.48
## - x5    1 324.14
## - x2    1 324.92
## - x1    1 336.11
```

```
## Stepwise Model Path
## Analysis of Deviance Table
##
## Initial
## mu
##  Model:
## y ~ x1 + x2 + x3 + x4 + x5 + x6
##
## Final
## mu
##  Model:
## y ~ x1 + x2 + x3 + x4 + x5
##
##
##   Step Df    Deviance Resid. Df Resid. Dev      AIC
## 1                           33   303.1602 319.1602
## 2 - x6  1 0.001715758        34   303.1619 317.1619
```

The above backward search procedure confirms that, if we want to include only linear additive terms in the model, the variable `x6` is not needed. The default penalty for the GAIC procedure is $k = 2$, i.e. a genuine original AIC selection procedure. Note that changing to a SBC the resulting model can be completely different as the following code is showing:

```
mod21<-stepGAIC(mod1, k=length(usair$y))
```

```
## Distribution parameter:  mu
## Start:  AIC= 631.16
##  y ~ x1 + x2 + x3 + x4 + x5 + x6
##
## . . .
## Step:  AIC= 455.04
##  y ~ x1
##
##        Df    AIC
## - x1    1 431.71
## <none>    455.04
##
## Step:  AIC= 431.71
##  y ~ 1
```

Here using SBC no explanatory variable is selected. (Note that we have only 41 observations and therefore any result should be treated with caution).

As an example of using the `scope` argument explicitly we consider whether two way interactions between the explanatory variables are needed in the model. The simplest model we are considering here is with only a constant, i.e. `lower= 1`, and the most complicated is the one with all two way interactions. The final model will be something between those two.

```
mod3<-stepGAIC(mod1, scope=list(lower=~1,upper=~(x1+x2+x3+x4+x5+x6)^2))
mod3$anov
```

```
## Distribution parameter:  mu
## Start:  AIC= 319.16
##  y ~ x1 + x2 + x3 + x4 + x5 + x6
##
## . . .
## Step:  AIC= 292.72
##  y ~ x1 + x2 + x3 + x4 + x5 + x6 + x4:x5 + x1:x6 + x4:x6 + x3:x4 +
##      x2:x4 + x2:x3 + x3:x6 + x2:x6
##
##         Df    AIC
## <none>     292.72
## + x1:x4  1 293.55
## + x1:x5  1 293.95
## + x2:x5  1 294.08
## - x2:x6  1 294.19
## + x5:x6  1 294.54
## + x3:x5  1 294.55
## + x1:x2  1 294.71
```

```
## + x1:x3  1 294.72
## - x1:x6  1 295.18
## - x3:x6  1 296.41
## - x2:x3  1 297.34
## - x3:x4  1 300.27
## - x2:x4  1 300.41
## - x4:x6  1 307.60
## - x4:x5  1 328.13
```

```
## Stepwise Model Path
## Analysis of Deviance Table
##
## Initial
## mu
##  Model:
## y ~ x1 + x2 + x3 + x4 + x5 + x6
##
## Final
## mu
##  Model:
## y ~ x1 + x2 + x3 + x4 + x5 + x6 + x4:x5 + x1:x6 + x4:x6 + x3:x4 +
##     x2:x4 + x2:x3 + x3:x6 + x2:x6
##
##
##      Step Df  Deviance Resid. Df Resid. Dev      AIC
## 1                             33   303.1602 319.1602
## 2 + x4:x5  1 14.086994        32   289.0732 307.0732
## 3 + x1:x6  1  8.133304        31   280.9399 300.9399
## 4 + x4:x6  1  4.786482        30   276.1534 298.1534
## 5 + x3:x4  1  2.082757        29   274.0706 298.0706
## 6 + x2:x4  1  4.460528        28   269.6101 295.6101
## 7 + x2:x3  1  2.886924        27   266.7232 294.7232
## 8 + x3:x6  1  2.532079        26   264.1911 294.1911
## 9 + x2:x6  1  3.468607        25   260.7225 292.7225
```

Model `mod3` is a rather complicated interaction model. [A simpler model could be selected by using a higher value of `k` rather the default `k=2`]. Note that the variable `x6` is included in the model `mod3` since higher interactions involving `x6` are selected in the model. More than two way interactions are not permitted for continuous variables which is the case in our example. A plot of the residuals of model `mod3` indicates possible heterogeneity in the variation of Y. We shall deal with this problem later. Fist we show how to select smoothing terms. In order to do that we first create a formula containing all the linear main effects and second order interactions plus smooth functions (using `pb()`) of the explanatory variables.

```
FORM1 <- as.formula( paste("~",paste("(x1+x2+x3+x4+x5+x6)^2+",
           paste(paste(paste("pb(", names(usair[-1]), sep=""),")", sep=""),
           collapse="+"), sep="" ), sep=""))
FORM1
```

```
## ~(x1 + x2 + x3 + x4 + x5 + x6)^2 + pb(x1) + pb(x2) + pb(x3) +
```

```
##     pb(x4) + pb(x5) + pb(x6)
```

We will use `FORM1` as an `upper` argument for `scope`

```
mod10<- stepGAIC(mod0, scope=list(lower=~1, upper=FORM1))
```

```
## Distribution parameter:  mu
## Start:  AIC= 353.71
##  y ~ 1
##
## . . .
##
##
## Step:  AIC= 304.29
##  y ~ pb(x4) + pb(x5) + x2 + x3 + pb(x1)
##
##                    Df    AIC
## <none>                304.29
## + pb(x2)  4.7610e-05 304.29
## + pb(x3)  5.3776e-05 304.29
## + pb(x6)  1.2605e+00 306.10
## + x6      1.2603e+00 306.10
## + x2:x3   9.9942e-01 306.30
## - x3      2.0294e+00 310.59
## - pb(x1) -1.4964e-01 312.04
## - pb(x5)  2.2641e+00 313.26
## - x2      1.9560e+00 315.40
## - pb(x4)  1.7994e+00 315.56
```

The idea here, is that we would like to check whether smoothing terms, linear terms or second order linear interactions are needed for modelling the μ. The resulting model:

```
pb(x4)+pb(x5)+x2+x3+pb(x1)
```

contains smooth functions for `x1`, `x4` and `x5` and linear terms for `x2` and `x3` but not interactions terms. Note however that by default, using `pb()`, the linear part of the explanatory variable is fitted separately, but its not recognise by the `stepGAIC()` function and therefore interactions involving smoothing terms never enter into consideration. This is a limitation of the `stepGAIC()` function which the user has to be aware. In the output above only the `x2:x3` interaction was tested since those are the two linear main effects fitted explicitly.

11.4.2 Selecting model for σ

We shall now try to include linear terms in the σ model. Note that with only 41 observations and with a reasonably complicated model μ, it **not** advisable to try smoothing terms for σ. Here we check whether including linear terms in the model for σ will improve model `mod1` which includes all linear terms, i.e. reduce AIC using the `stepGAIC` function.

```
mod4 <- stepGAIC(mod1, parameter="sigma", scope=~x1+x2+x3+x4+x5+x6)
```

```
## Distribution parameter:  sigma
## Start:  AIC= 319.16
##  ~1
##
## . . .
## Step:  AIC= 314.21
##  ~x5 + x1
##
##        Df    AIC
## <none>    314.21
## + x3    1 314.57
## + x2    1 315.58
## + x6    1 315.59
## + x4    1 316.04
## - x1    1 317.32
## - x5    1 319.29
```

According to criterion AIC the model needs `x1+x5` in the formula for σ. A method which selects terms for all the parameter of the distribution is described next.

11.5 Strategy A: the `stepGAICAll.A()` function

Strategies A and B are strategies for selecting additive terms using a GAIC for all the parameters of the distribution of the response variable. Strategy A can be described as follows:

For a fixed distribution:

1. Use a forward GAIC selection procedure to select an appropriate model for μ, with σ, ν and τ fitted as constants.
2. Given the model for μ obtained in (1) and for ν and τ fitted as constants, use a forward selection procedure to select an appropriate model for σ.
3. Given the models for μ and σ obtained in (1) and (2) respectively and with τ fitted as constant, use a forward selection procedure to select an appropriate model for ν.
4. Given the models for μ, σ and ν obtained in (1), (2) and (3) respectively, use a forward selection procedure to select an appropriate model for τ.
5. Given the models for μ, σ and τ obtained in (1), (2) and (4) respectively, use a backward selection procedure to select appropriate model for ν,
6. Given the models for μ, ν and τ obtained in (1), (5) and (4) respectively, use a backward selection procedure to select appropriate model for σ.
7. Given the models for σ, ν and τ obtained in (6), (5) and (4) respectively, use a backward selection procedure to select an appropriate model for μ and then stop.

The final model will contain different subset of terms (not necessarily the same terms) for each μ, σ, ν and τ. This is illustrated in Table 11.3 showing for example, that among all the available variables $x_1, x_2 \ldots, x_6$, the variable x_1 was chosen only for μ and ν but not for σ or τ.

	x_1	x_2	x_3	x_4	x_5	x_6
μ	✓		✓	✓		✓
σ			✓	✓		
ν	✓		✓			
τ				✓		

Table 11.3: Showing a possible result from a selection of variables using strategy A. Among all available variables $x_1, x_2 \ldots, x_6$, some were chosen for μ, some for σ, some for ν and some for τ.

The function to perform the strategy A is `stepGAICAll.A()` and has the following arguments.

`object`	an gamlss object which is used as the initial model in the stepwise search.
`scope`	the scope should be a list with elements `lower` and `upper` containing formulae.
`sigma.scope`	the scope of σ if different from `scope`
`nu.scope`	the scope of ν if different from `scope`
`tau.scope`	the scope of τ if different from `scope`
`mu.try`	the default value is TRUE and can be set to FALSE if no model selection for μ is needed
`sigma.try`	the default value is TRUE, can be set to FALSE if no model selection for σ is needed
`nu.try`	the default value is TRUE, can be set to FALSE if no model selection for ν is needed
`tau.try`	the default value is TRUE, can be set to FALSE if no model selection for τ is needed

Next we use the function `stepGAICAll.A()` with penalty `k=log(41)`, i.e. SBC, to select linear terms for both μ and σ.

```
m1 <- gamlss(y~1, data=usair, famly=GA, trace=FALSE)
m2<- stepGAICAll.A(m1, scope=list(lower=~1, upper=~x1+x2+x3+x4+x5+x6))
```

```
## ----------------------------------------------------
## Distribution parameter:  mu
## Start:  AIC= 381.54
##  y ~ 1
## . . .
## + x5    1 354.66
## ----------------------------------------------------
## Distribution parameter:  mu
## Start:  AIC= 351.37
##  y ~ x2 + x3
```

```
##
##        Df    AIC
## <none>    351.37
## - x3    1 357.91
## - x2    1 374.47
## ----------------------------------------------------
```

```
m2
```

```
##
## Family:  c("NO", "Normal")
## Fitting method: RS()
##
## Call:
## gamlss(formula = y ~ x2 + x3, sigma.formula = ~x4 + x3, data = usair,
##     famly = GA, trace = FALSE)
##
## Mu Coefficients:
## (Intercept)           x2           x3
##    23.69223      0.06678     -0.04035
## Sigma Coefficients:
## (Intercept)           x4           x3
##   0.8881979    0.2200108   -0.0006106
##
##  Degrees of Freedom for the fit: 6 Residual Deg. of Freedom   35
## Global Deviance:     329.089
##             AIC:     341.089
##             SBC:     351.37
```

The the parameters μ the linear terms `x2` and `x3` were selected while for σ, `x3` and `x4`.

Note that smoothing terms can be included but for this specific data with only 41 observations can hazardous. For example the following code would result to some errors due to the fact that the sigma models had failed:

```
m3 <- stepGAICAll.A(m1, scope=list(lower=~1,
          upper=~pb(x1)+pb(x2)+pb(x3)+pb(x4)+pb(x5)+pb(x6)), k=log(41))
```

Also note that during the selection procedure some models can fail but the selection is usually carried on until the end.

11.6 Strategy B: the `stepGAICAll.B()` function

This strategy forces all the distributions parameters to have the same term if selected. That is, if a terms from $\mathcal{X}$ is selected it is included in the predictor of all the parameters. The inclusion using GAIC can be done using forward, backward or stepwise procedure. Table 11.4 shows a possible result from strategy B.

The function to perform the strategy B is `stepGAICAll.B()` which uses repeatedly the functions `add1All()` and `drop1All()`, It has the following arguments.

	x_1	x_2	x_3	x_4	x_5	x_6
μ	✓		✓			✓
σ	✓		✓			✓
ν	✓		✓			✓
τ	✓		✓			✓

Table 11.4: Showing a possible result from a selection of variables using strategy B. Among all available variables $x_1, x_2 \ldots, x_6$, the selected terms are selected for all the parameters of the distribution.

`object` an gamlss object which is used as the initial model in the stepwise search.

`scope` the scope should be a list with elements `lower` and `upper` contain formulae.

`direction` the mode of the stepwise search, which can be one of `both`, `backward`, or `forward`, with a default of `both`. If the scope argument is missing the default for direction is `backward`

`trace` if positive, information is printed during the running of stepAIC. Larger values may give more information on the fitting process.

`keep` a filter function whose input is a fitted model object and the associated 'AIC' statistic, and whose output is arbitrary. Typically 'keep' will select a subset of the components of the object and return them. The default is not to keep anything.

`steps` the maximum number of steps to be considered. The default is 1000 (essentially as many as required). It is typically used to stop the process early.

`scale` scale is not used in **gamlss**

`k` the multiple of the number of degrees of freedom used for the penalty.

In addition also has the parallel computations arguments `parallel`,`ncpus` and `cl` as defined in Section 11.3. Here is an example of how the function can be used:

```
m4<- stepGAICAll.B(m1, scope=list(lower=~1, upper=~x1+x2+x3+x4+x5+x6),
                   k=log(41))
```

```
## Start:  AIC= 381.54
##  y ~ 1
##
##        Df    AIC
## . . .
## Step:  AIC= 350.55
##  y ~ x1 + x2 + x3 + x4 + x5
##
##        Df    AIC
## <none>    350.55
## - x3    2 351.26
## - x5    2 351.84
## - x2    2 353.71
```

```
## + x6    2 353.88
## - x4    2 358.62
## - x1    2 360.50
```

```
m4
```

```
##
## Family:  c("NO", "Normal")
## Fitting method: RS()
##
## Call:
## gamlss(formula = y ~ x1 + x2 + x3 + x4 + x5, sigma.formula = ~x1 +
##     x2 + x3 + x4 + x5, data = usair, famly = GA, trace = FALSE)
##
## Mu Coefficients:
## (Intercept)           x1           x2           x3           x4
##   116.08769     -1.09160      0.04495     -0.01692     -5.25258
##          x5
##     0.31086
## Sigma Coefficients:
## (Intercept)           x1           x2           x3           x4
##   6.0343687   -0.0815922    0.0008033   -0.0014193    0.0278921
##          x5
##   0.0287338
##
##  Degrees of Freedom for the fit: 12 Residual Deg. of Freedom   29
## Global Deviance:     305.988
##             AIC:     329.988
##             SBC:     350.551
```

The variables `x1`,`x2`,`x3` `x4` and `x5` where selected for both μ and σ.

11.7 Boosting

11.8 K-fold Cross Validation

Cross validation modelling can be achieved with GAMLSS using the function `gamlssCV()`. Models fitted with `gamlssCV()` can be compared using the function `CV()`. The first few arguments of `gamlssCV()` are similar to the `gamlss()` function arguments. i.e. `formula`, `sigma.formula`, ..., `family` etc. Also the parallel computations arguments `parallel`, `ncpus` and `cl` explained in Section 11.3 are available for speeding the procedure. The k-fold cross validation can be specified either by defining a factor in argument `rand` with levels the different cross validation data sets or by specified the argument `K.fold`. If the latest argument is chosen the different data sets will be created randomly. In this case you may wish to use the `set.seed` for repeatability of your results.

Here we use the `abdom` data first described in Chapter 4 to test using k-fold cross validation

whether we should use normal, `NO`, logistic, `LO`, or t, `TF`, distribution for modelling the response variable. To demonstrate the use of `parallel` argument we fit the three models using the `"no"`, `"multicore"` and `"snow"` arguments respectively.

```
#--------------------------------------------------------------------------------
#--------------------------------------------------------------------------------
# function gamlssCV
#--------------------------------------------------------------------------------
set.seed(123)
rand1 <- sample (10 , 610, replace=TRUE)
# detacting how many cores exist in the machine
nC <- detectCores()
#--------------------------------------------------------------------------------
```

```
# no parallel
g1 <- gamlssCV(y~pb(x,df=2), sigma.formula=~pb(x,df=1), data=abdom,
               family=NO, rand=rand1,parallel = "no", ncpus = nC )
```

```
## fold 1
## new prediction
## new prediction
## . . .
## new prediction
## fold 10
## new prediction
## new prediction
```

```
# using multicore
g2 <- gamlssCV(y~pb(x,df=2), sigma.formula=~pb(x,df=1), data=abdom,
               family=LO, rand=rand1,parallel = "multicore", ncpus = nC )
# using snow
g3 <- gamlssCV(y~pb(x,df=2), sigma.formula=~pb(x,df=1), data=abdom,
               family=TF, rand=rand1,parallel = "snow", ncpus = nC )
CV(g1,g2,g3)
```

```
##    val[o.val]
## g2   4804.336
## g3   4804.564
## g1   4814.716
```

From the output the logistic distribution, `LO`, seems to be selected using the 10-fold cross validation.

11.9 Validation, and test data

11.9.1 The `gamlssVGD()` and `VGD()` functions

Fitting a model in a training data set and validating the model in a different validation/test data set can be achieved within GAMLSS using the function `gamlssVGD()`. Models fitted

with `gamlssVGD()` can be compared between them using the function `VGD()`. The function `gamlssVGD()` works similar to the function `gamlssCV()` described above in Section 11.8. The main arguments for the function are the `gamlss` arguments. Additional arguments are `rand` and `newdata`. Those two arguments determine how the spilt into the two data sets (training and validation) is done. If the data are already split into two `data.frames` then the `data` and `newdata` arguments can be used to specify the training and the validation/test data set respectively. If on the other hand there is a single data set then the argument `rand` can be used to define which part of the data will be used for training and which for validation/test. We demonstrate the difference between those two approaches below. First we generate a factor which splits the 610 observations of the data `abdom` into two groups containing 60% and 40% approximately. Then we fit three different models using the normal, `NO`, the logistic, `LO`, and the t, `TF`, distributions respectively and compare the validation global deviance using the function `TGD()`.

```
# generate the random split of the data
rand <- sample(2, 610, replace=TRUE, prob=c(0.6,0.4))
# the proportions in the sample
table(rand)/610
```

```
## rand
##         1         2
## 0.6311475 0.3688525
```

```
#------------------------------------------------------------------------------
# using the argument rand
v1 <- gamlssVGD(y~pb(x,df=2),sigma.formula=~pb(x,df=1), data=abdom, family=NO,
              rand=rand)
```

```
## new prediction
## new prediction
```

```
v2 <- gamlssVGD(y~pb(x,df=2),sigma.formula=~pb(x,df=1), data=abdom, family=LO,
              rand=rand)
```

```
## new prediction
## new prediction
```

```
v3 <- gamlssVGD(y~pb(x,df=2),sigma.formula=~pb(x,df=1), data=abdom, family=TF,
              rand=rand)
```

```
## new prediction
## new prediction
```

```
VGD(v1,v2,v3)
```

```
##    val[o.val]
## v2   1765.934
## v3   1769.929
## v1   1775.608
```

Next, we repeat the same analysis but this time the data are split into two sets in advance.

```
#------------------------------------------------------------------------------
# using the two different data set
```

```
# create training and validation data sets
olddata<-abdom[rand==1,] # training data
newdata<-abdom[rand==2,] # validation data
v11 <- gamlssVGD(y~pb(x,df=2),sigma.formula=~pb(x,df=1), data=olddata,
                  family=NO, newdata=newdata)

## new prediction
## new prediction

v12 <- gamlssVGD(y~pb(x,df=2),sigma.formula=~pb(x,df=1), data=olddata,
                 family=LO, newdata=newdata)

## new prediction
## new prediction

v13 <- gamlssVGD(y~pb(x,df=2),sigma.formula=~pb(x,df=1), data=olddata,
                 family=TF, newdata=newdata)

## new prediction
## new prediction

VGD(v11,v12,v13)

##     val[o.val]
## v12   1765.934
## v13   1769.929
## v11   1775.608
```

The logistic distribution model is supported by the data.

11.9.2 The `getTGD()` and `TGD()` functions

The function `getTGD()` and `TGD()` are doing similar job to the functions `gamlssVGD()` and `VGD()` respectively with the difference that they assumed that the models involved have been already fitted using the training data set and now we only need to compare how well they fit to the validation/test data set. That is, given the fitted models we would like to compare them using the global deviance evaluated at the validation/test data set, which is defined by the argument `newdata`.

```
#-------------------------------------------------------------------------------
# function getTGD
#-------------------------------------------------------------------------------
# fit gamlss models first
g1 <- gamlss(y~pb(x,df=2),sigma.formula=~pb(x,df=1), data=olddata, family=NO,
             trace=FALSE)
g2 <- gamlss(y~pb(x,df=2),sigma.formula=~pb(x,df=1), data=olddata, family=LO,
               trace=FALSE)
g3 <- gamlss(y~pb(x,df=2),sigma.formula=~pb(x,df=1), data=olddata, family=TF,
              trace=FALSE)
# and then use
gg1 <-getTGD(g1, newdata=newdata)
```

```
## new prediction
## new prediction

gg2 <-getTGD(g2, newdata=newdata)

## new prediction
## new prediction

gg3 <-getTGD(g3, newdata=newdata)

## new prediction
## new prediction

TGD(gg1,gg2,gg3)

##     val[o.val]
## gg2   1765.934
## gg3   1769.929
## gg1   1775.608
```

11.9.3 The stepTGD() function

The function `stepTGD()` behaves similar to the `stepGAIC()` function but it uses the validation/test global deviance instead of GAIC as the selection criterion. The functions `add1TGD()`, `drop1TGD()` are used by `stepTGD()` in the same way that `addterm()`, `dropterm()` are used by `stepGAIC()`. The arguments of the function `stepTGD()` are similar to the ones in `stepGAIC()` with the addition of the argument `newdata` which is expecting the validation/test data set.

To demonstrate the use of the function we will use the Munich rent data first used in Chapter **??**. We split the data into training and validation data sets and use the training data to fit a null model `v0` and a more complicated model with four terms, `v1`. We then use those two models to demonstrate the `drop1TGD()` and `add1TGD()` functions.

```
# the data
set.seed(123)
rand <- sample(2, dim(rent)[1], replace=TRUE, prob=c(0.6,0.4))
# the proportions in the sample
table(rand)/dim(rent)[1]

## rand
##         1         2
## 0.6094464 0.3905536

oldrent<-rent[rand==1,] # training set
newrent<-rent[rand==2,] # validation set
# null model
v0 <- gamlss(R~1, data=oldrent, family=GA, trace=FALSE)
# complete model
v1 <- gamlss(R~pb(Fl)+pb(A)+H+loc, sigma.fo=~pb(Fl)+pb(A)+H+loc,
             data=oldrent, family=GA, trace=FALSE)
# drop1TGDP
nC <- detectCores()
```

```
(v2<- drop1TGD(v1, newdata=newrent,  parallel="snow", ncpus=nC))
```

```
## new prediction
## new prediction
## Single term deletions for
## mu
##
## Model:
## R ~ pb(Fl) + pb(A) + H + loc
##            Df   TGD
## <none>        10852
## pb(Fl) 2.0659 11081
## pb(A)  4.1375 10882
## H      1.3579 10892
## loc    2.4285 10876
```

```
# add1TGDP
(v3<- add1TGD(v0, scope=~pb(Fl)+pb(A)+H+loc,newdata=newrent,
                        parallel="snow", ncpus=nC))
```

```
## Single term additions for
## mu
##
## Model:
## R ~ 1
##            Df   TGD
## <none>        11242
## pb(Fl) 1.9378 11000
## pb(A)  3.9365 11225
## H      1.0000 11164
## loc    2.0000 11213
```

To demonstrate the `stepTGD()` function we start from the null model:

```
# stepTGD
v4<- stepTGD(v0, scope=~pb(Fl)+pb(A)+H+loc,newdata=newrent,
                          parallel="snow", ncpus=nC)
```

```
## Distribution parameter:  mu
## Start:  TGD= 11241.88
##  R ~ 1
##
## . . .
## Step:  TGD= 10874.09
##  R ~ pb(Fl) + H + loc + pb(A)
##
## new prediction
## new prediction
##              Df   TGD
## <none>          10874
## - pb(A)  3.6739 10894
```

```
## - loc    1.8784 10906
## - H      1.6700 10937
## - pb(Fl) 1.6353 11113
```

Note that the results above shown that all four terms are needed in modelling the μ parameter and therefore no reduction of variables is required.

11.10 The `find.hyper()` function

Estimation of the smoothing parameters has been discussed in Section 3.3 of Chapter 3 and also in the beginning of this Chapter. We have distinguished the methods used to two main categories the global, when the methods are applied outside the GAMLSS algorithm, and the local when are applied within. Local methods have been discussed in both Chapters 3 and 9. Here we focus on the function `find.hyper()` which is a global method for estimating smoothing parameters and appears to work well in searching for the optimum degrees of freedom for smoothing and/or non-linear parameters (e.g. a power parameter ξ used to transform x to x^{ξ}). The function repetitively fits GAMLSS models and uses the **R** function `optim()` to minimizes the generalized Akaike information criterion (GAIC) for a given penalty `k` specified by the user. For large data sets the function is very slow compared to local estimation methods and for this reason is hardly been used recently. Here the results of the function are compared with local methods of estimation of the smoothing parameters.

The arguments of the function `find.hyper()` are:

`model` this is a quoted ((quote())) GAMLSS model in which the required hyperparameters are denoted by `p[number]`, e.g.
`quote(gamlss(y~cs(x,df=p[1]),sigma.fo=~ cs(x,df=p[2]),data=abdom))`

`parameters` the starting parameter values in the search for the optimum hyperparameters and/or non-linear parameters, e.g. `parameters=c(3,3)`

`other` this is used to optimize non-linear parameter(s), for example a transformation of the explanatory variable of the kind $x^{p[3]}$, e.g. `others=quote(nx<-x^p[3])` where nx is now in the model formula

`k` specifies the penalty in the GAIC, (the default is 2) e.g. `penalty=3`

`steps` the steps in the parameter(s) taken during the optimization procedure (see for example the `ndeps` option in the control function for `optim()`), by default set to 0.1 for all hyper parameters and non-linear parameters

`lower` the lower bounds on the permissible values of the parameters e.g. for two parameters `lower=c(1,1)`. This does not apply if a method other than the default method "L-BFGS-B" is used

`upper` the upper bounds on the permissible values of the parameters e.g. for two parameters `upper=c(30,10)`. This does not apply if a method other than the default method "L-BFGS-B" is used

`method` the method used in `optim()` to numerically minimize the GAIC over the

hyperparameters and/or non-linear parameters. By default this is "L-BFGS-B" to allow box-restriction on the parameters

... this can be used for extra arguments in the `control` argument of the `R` function `optim()`

The function `find.hyper()` returns the same output as the **R** function `optim()`.

In the following example we compare the local and global estimation of the smoothing parameters using the `abdom` data. Both models for μ and σ are fitted sing a non-parametric P-spline. We first use the three local methods for estimating the smoothing parameter `"ML"`, `"GCV"` and `"GAIC"`.

```
# fitting the model with pb()
a1 <- gamlss(y ~ pb(x), sigma.fo=~pb(x), data = abdom, family = LO,
             trace=FALSE)
a2 <- gamlss(y ~ pb(x, method="GCV"), sigma.fo=~pb(x, method="GCV"),
             data = abdom, family = LO, trace=FALSE)
a3 <- gamlss(y ~ pb(x, method="GAIC"), sigma.fo=~pb(x, method="GAIC"),
              data = abdom, family = LO, trace=FALSE)
# the effective degrees of freedom used
edfAll(a1);edfAll(a2);edfAll(a3)
```

```
## $mu
##    pb(x)
## 5.796263
##
## $sigma
##    pb(x)
## 2.001641
## $mu
## pb(x, method = "GCV")
##              4.842148
##
## $sigma
## pb(x, method = "GCV")
##              2.600804
## $mu
## pb(x, method = "GAIC")
##               2.000007
##
## $sigma
## pb(x, method = "GAIC")
##               2.001502
```

Now we will use the global GAIC method to find the degrees of freedom. First we have to declare the model using the `quote` **R** function. For each hyper-parameter to be estimated we put `p[.]` with the appropriate number in the square brackets. The function `find.hyper()` minimises GAIC with `k=2` by default. The initial degrees of freedom parameters, `p`, for the search is set to 3 (i.e. `parameters=c(3,3)`), the minimum value for p for the search is set to 0 (i.e. lower=c(0,0)) and the steps in $p[1]$ used within the `optim()` search to 0.1 (i.e. `steps=c(0.1)`). The default

method used by `optim()` within `find.hyper()` is the "L-BFGS-B" procedure which starts with the initial parameter value(s), changes each parameter in turn by ± step for that parameter, and then jumps to new value(s) for the set of parameter(s). This is repeated until convergence. See the help on the **R** function `optim()` for details. Note that `df` as defined in `pb()` are the effective degrees of freedom on top of the constant and linear, so `df=0` corresponds to a linear fit.

```
mod1 <- quote(gamlss(y ~ pb(x, df = p[1]), sigma.fo=~pb(x, df=p[2]),
             family = LO, data = abdom, trace = FALSE))
```

```
op <- find.hyper(model = mod1, par = c(3,3), lower = c(0,0), steps = c(0.1),
      trace = FALSE)
```

```
## par 3 3 crit= 4798.775 with pen= 2
## par 3.1 3 crit= 4798.599 with pen= 2
## . . .
## par 3.711021 0 crit= 4795.093 with pen= 2
## par 3.710816 0 crit= 4795.093 with pen= 2
## par 3.810816 0 crit= 4795.101 with pen= 2
## par 3.610816 0 crit= 4795.101 with pen= 2
## par 3.710816 0.1 crit= 4795.093 with pen= 2
## par 3.710816 0 crit= 4795.093 with pen= 2
```

```
op
```

```
## $par
## [1] 3.710816 0.000000
##
## $value
## [1] 4795.093
##
## $counts
## function gradient
##       10       10
##
## $convergence
## [1] 0
##
## $message
## [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

The resulting extra degrees of freedom are 3.710816 for μ and 0 for σ corresponding to total effective degrees of freedom 5.710816 for μ and 2 for σ. Those are close to the results obtained using the local `"ML"` method.

Chapter 12

Diagnostics

This chapter provides:

1. provides the definition of normalised (randomised) quantile residuals and
2. other diagnostic tools based on residuals, such as the worm plots, `wp()`, and Q-Statistics, `Q.stats()`, functions

This chapter is important for understanding the tools for checking the adequacy of a GAMLSS model.

12.1 Introduction

In the simple linear regression model $y_i = \beta_0 + \beta_1 x_i + e_i$ we defined the residuals as the difference between the observed and the fitted values $\hat{\varepsilon}_i = y_i - \hat{y}_i$ where $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$ for $i = 1, 2, \ldots, n$. Sometimes $\hat{\varepsilon}_i$'s are called the *raw* residuals to distinguish them from the *standardised residuals* which are defined as $(y_i - \hat{y}_i)/\hat{\sigma}\sqrt{(1-h_{ii})}$ where h_{ii} are the diagonal values of the hat matrix. The problem with raw residuals is that they are difficult to generalize to different distributions other than the normal. For example, within the generalised linear model literature the *deviance residuals* $r_i^d = \text{sign}\,(y_i - \hat{\mu}_i)\,/\sqrt{d_i}$ where $d_i = -2\log(L_i^c/L_i^s)$ [1] or the *Pearson's residuals* $r_i^P = (y_i - \hat{\mu}_i)\,/se(\hat{\mu}_i)$ are often used. Unfortunately the deviance residuals are not well defined with multiple parameters for the distribution of y, while the Pearson residuals can be far from a normal distribution and also are not appropriate for highly skew or kurtotic data. Therefore for GAMLSS models we use the *normalised (randomised) quantile residuals*, Dunn and Smyth [1996], and we refered to as 'residuals' through this book.

Section 12.2 introduces the normalised quantile residuals for continuous response variables. For discrete response variables the normalised quantile residuals have to be randomised so we call them normalised *randomised* quantile residuals. Section 12.3 describes the `plot()` function of GAMLSS. Section 12.4 describes the worm plot function, `wp()`. The Q-statistics function `Q.stats()` is considered in section 12.5 while the `rqres.plot()` function in section 12.6.

[1] L^c represent the likelihood for the current model and L^s from the saturated model that is, when $\hat{\mu}_i = y_i$, McCullagh and Nelder [1989].

12.2 Normalised (randomised) quantile residuals

This section first introduces the normalised quantile residuals and then explains how they can be used within the package.

The main advantage of the normalised (randomised) quantile residuals is that, whatever the distribution of the response variable their true values r_i, $i = 1, 2, \ldots, n$ always have a standard normal distribution given the assumption that the model is correct. Since within the statistical literature checking the normality assumption is well established, the normalised (randomised) quantile residuals provide us with an easy way to check the adequacy of a GAMLSS fitted model.

Given that the distribution $f(y; \boldsymbol{\theta})$ is fitted to observations y_i for $i = 1, 2, \ldots, n$, the fitted normalised (randomised) quantile residuals, Dunn and Smyth [1996], are given by $\hat{r}_i = \Phi^{-1}(\hat{u}_i)$, where Φ^{-1} is the inverse cumulative distribution function of a standard normal variable. The $\hat{u}_i$'s are quantile residuals defined differently for continuous and discrete response variables.

If y is an observation from a continuous response variable then let $u = F(y|\boldsymbol{\theta})$ and $\hat{u} = F(y|\hat{\boldsymbol{\theta}})$ be the model and fitted cumulative distribution functions respectively. The process is described diagrammatically in Figure 12.1. The top plot shows the probability density function for a specific observation y. The middle plot shows how, using the cumulative distribution function, the observation y is mapped onto u. If the model is correctly specified u has a uniform distribution between zero and one. The u's are referred to the econometric literature as PIT (probability integral transform) residuals. In the bottom figure u is transformed into a *z-score*, r, using $r = \Phi^{-1}(u)$, the inverse cumulative distribution function of a standard normal variable, so r will have a standard normal distribution. Note that $r = \Phi^{-1}[F(y|\boldsymbol{\theta})]$. Similarly $\hat{u}$ is transformed to $\hat{r}$ by $\hat{r} = \Phi^{-1}(\hat{u}) = \Phi^{-1}\left[F(y|\hat{\boldsymbol{\theta}})\right]$ and $\hat{r}$ has an approximate standard normal distribution. (Note that the normalised quantile residual r is the z-score corresponding to observation y based on its distribution). If y is an observation from a discrete integer response variable then u is a random value from the uniform distribution on the interval $[u_1, u_2] = [F(y-1|\boldsymbol{\theta}), F(y|\boldsymbol{\theta})]$ and $\hat{u}$ is a random value from a uniform distribution on $[\hat{u}_1, \hat{u}_2] = \left[F(y-1|\hat{\boldsymbol{\theta}}), F(y|\hat{\boldsymbol{\theta}})\right]$. The process is described in Figure 12.2. For a given discrete probability function (top graph), the observed y value is transformed into an interval (u_1, u_2) (the shaded strip in middle plot). Then u is selected randomly from (u_1, u_2) and is transformed into the (randomised) z-score, $r = \Phi^{-1}(u)$, (see the bottom graph). Hence, r has exactly a standard normal distribution if the model is correct. Similarly, using the fitted cumulative distribution function, y is transformed to $\hat{u}$, randomly chosen from $(\hat{u}_1, \hat{u}_2)$, and then transformed to $\hat{r} = \Phi^{-1}(\hat{u})$ and $\hat{r}$ has an approximate standard normal distribution.

The randomisation of quantile residuals is also appropriate for interval or censored response variables. For example, for a right censored continuous response, $\hat{u}$ is defined as a random value from a uniform distribution on the interval $\left[F(y|\hat{\boldsymbol{\theta}}), 1\right]$.

Note that, when randomisation is used, several randomised sets of residuals (or a median set from them) should be studied before a decision about the adequacy of model $\mathcal{M}$ is taken. Next codes show how to create the Figures 12.1 and 12.2 that gives a description of how a (normalised quantile) residual r is obtained for continuous and discrete a distribution, respectively.

The normalised (randomised) quantile residuals can be obtained in the **gamlss** package using

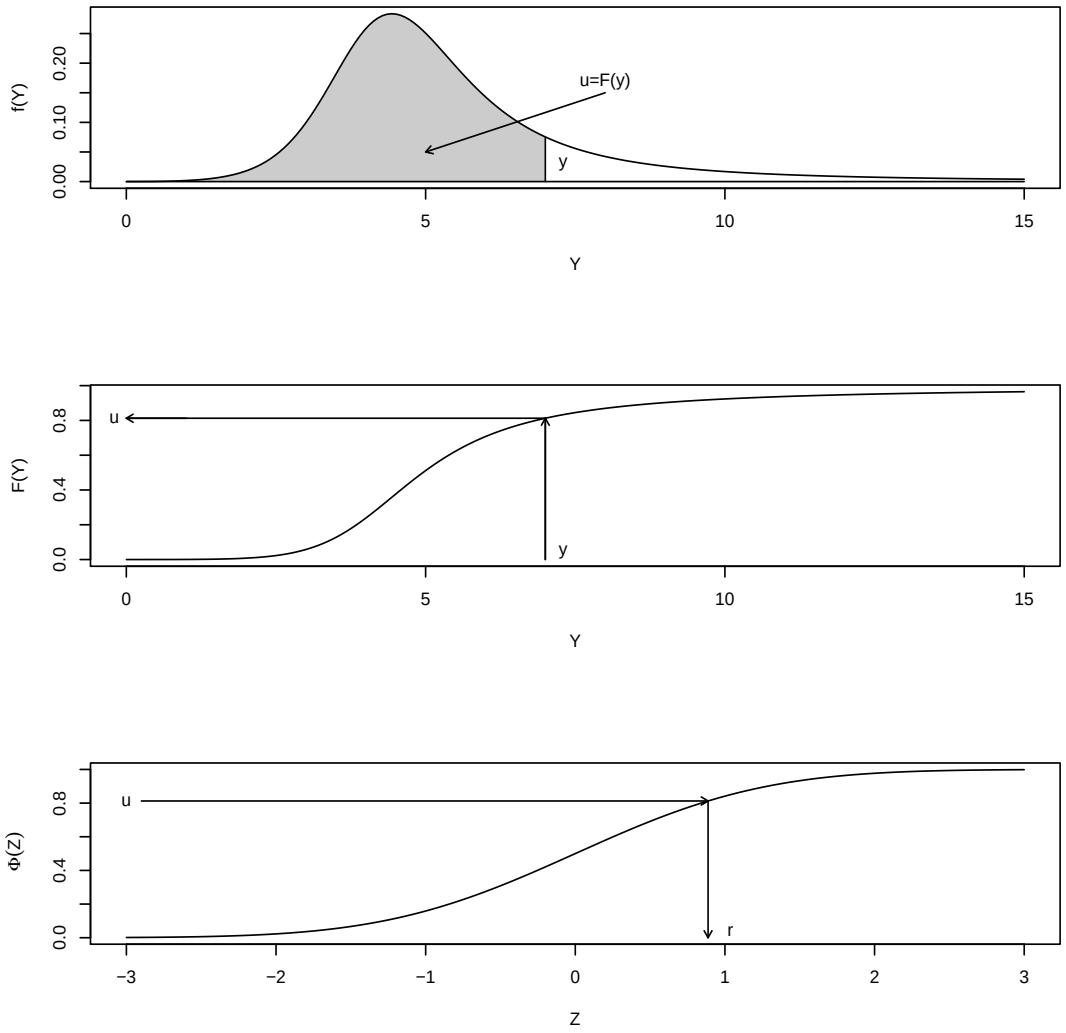


Figure 12.1: A description of how a (normalised quantile) residual r is obtained for continuous a distribution. The functions plotted are the model probability density function $f(y)$, the cumulative distribution function $F(y)$ and cumulative distribution function of a standard normal random variable $\Phi(z)$, using which y is transformed to u and then from u to r. The residual r is the z-score for the specific observation and has a standard normal distribution if the model is correct.

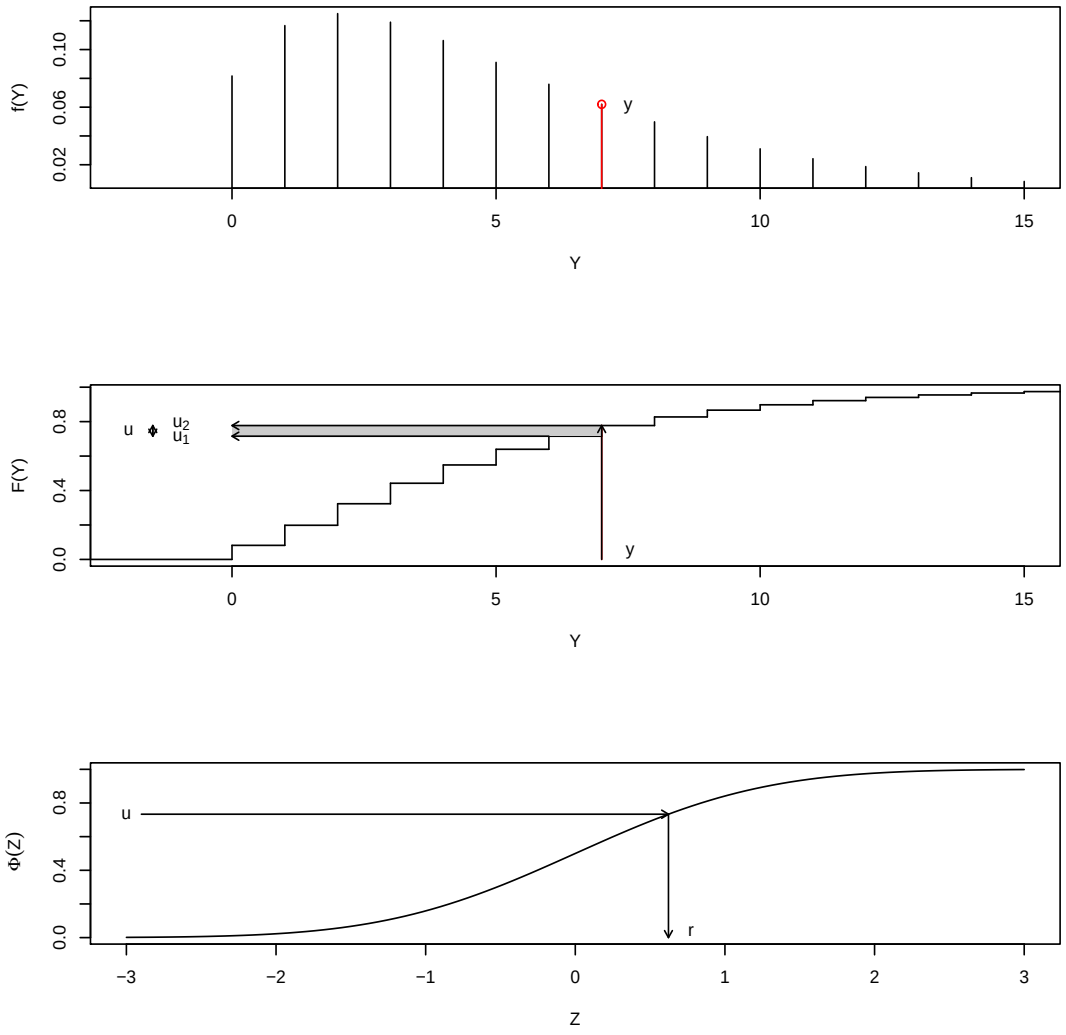


Figure 12.2: A description of how a (normalised randomised quantile) residual r is obtained for a discrete distribution. The observed y is transformed to u, a random value between u_1 and u_2, then u is transformed to r. The residual r is a z-score for the specific observation and has a standard normal distribution if the model is correct.

the function resid(). There are several other functions in the package using the normalised (randomised) quantile residuals.

- the function plot() is for general residual checking
- the worm plot function wp() which can be used to identify whether the fitted distribution is adequate either overall or within non-overlapping ranges of either one or two explanatory variables,
- the Q statistics function Q.stats for detecting the residuals are "significantly" different from a normal distribution in their mean, variance, skewness and kurtosis [and more potentially which distribution parameters of the model failed to fit adequately] in which ranges of an explanatory variable
- the function rqres.plot() designed for repeated randomisation of the residuals (when the response variable is not continuous).

All the above functions are explained below.

12.3 The plot() function

The full name of this function is plot.gamlss() but since it is a method function in **R** () it can be called using just plot() provided its first argument is a fitted gamlss object. The function plot() produces four plots for checking the normalised (randomised) quantile residuals defined in section 12.2 of a fitted gamlss object. Randomisation is performed for discrete and mixed response variables and also for interval or censored data. The four plots are

- residuals against the fitted values of the μ parameter
- residuals against an index or a specified x-variable
- a kernel density estimate of the residuals
- a QQ-normal plot of the residuals

When randomisation is performed (e.g. in the discrete distribution families) it is advisable to be used in conjunction with the function rqres.plot described in Section 12.6.

The arguments of the plot.gamlss() function are

x	a gamlss fitted object
xvar	an explanatory variable to plot the residuals against. By default the index 1:N is plotted, where N is the total number of observations.
parameters	this option can be used to change the default parameters in the plotting. The current default parameters are par(mfrow=c(2,2), mar=par("mar")+ c(0,1,0,0), col.axis = "blue4", col = "darkgreen", bg = "beige"). These parameters are not appropriate, when someone wishes to include the plot into a document. We have found that the option parameters= par(mfrow = c(2,2), mar = par("mar") + c(0,1,0 col.main = "blue4", col.lab = "blue4", pch = "+", cex = .45, cex.lab = 1.2, cex.axis = 1, cex.main = 1.2) gives reasonable plots for printed documents.

`ts` set this to TRUE if ACF and PACF plots of the residuals are required. This option is appropriate if the response variable is a time series. The ACF and PACF then replace the first two of the four plots listed above.

`summaries` set this to FALSE if no summary statistics of the residuals are required. By default the function `plot.gamlss()` produces some summary statistics for the (normalised randomised quantile) residuals.

Here is an example of how to used the plot function using the abdominal circumference data:

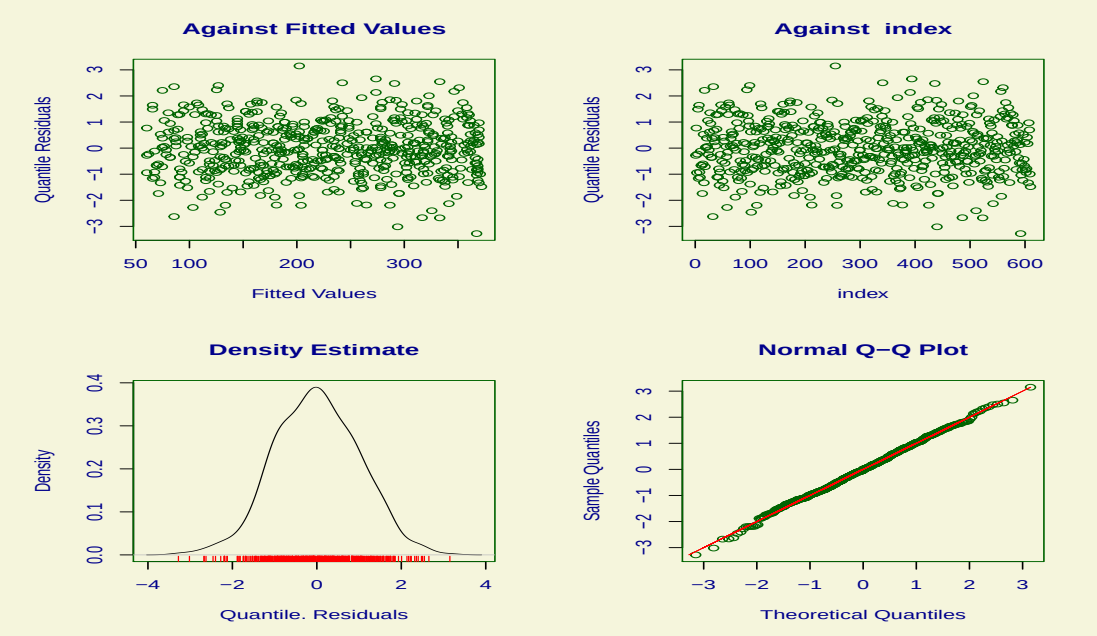
```
data(abdom)
abd10<-gamlss(y~pb(x),sigma.fo=~pb(x,df=1),data=abdom,family=BCT)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 4774.464
## . . .
## GAMLSS-RS iteration 7: Global Deviance = 4773.399
```

Figure 12.3

```
plot(abd10)
```

```
## ******************************************************************
##        Summary of the Quantile Residuals
##                             mean   =  0.0009096
##                         variance   =  1.002
##                 coef. of skewness  =  -0.008444
##                 coef. of kurtosis  =  2.993
## Filliben correlation coefficient   =  0.9993
## ******************************************************************
```



R code on page 292

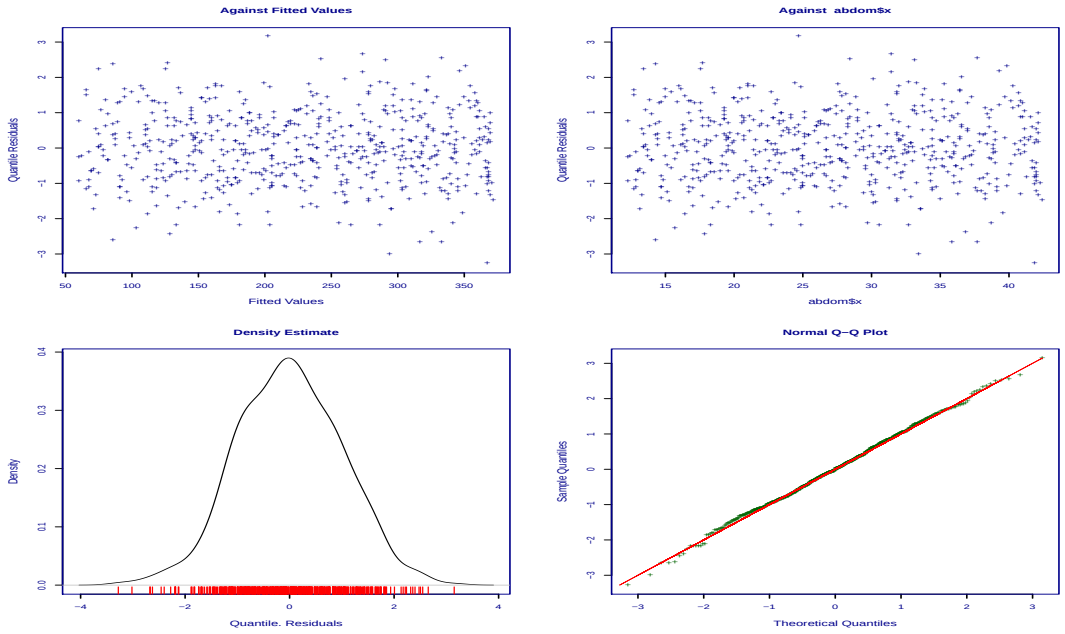
Figure 12.3: Residual plots from the BCT model `abd10`

The resulting plot is shown in Figure 12.3 Note that the the (normalised quantile) residuals of this model behave well, e.g. their mean is nearly zero, their variance nearly one, their coefficient of skewness near zero and their coefficient of kurtosis is near 3. The residuals are approximately normally distributed as they should be for an adequate model.

Let us now use some of the options. Here we use the option `xvar` to change the top right hand plot so the plot shows the residuals against age (`abdom$x`) instead of the index. Note though that this makes very little difference in the plot since age is already ordered. Also we change the plotting parameters values. The plot is shown in Figure 12.4.

```
newpar<-par(mfrow=c(2,2), mar=par("mar")+c(0,1,0,0), col.axis="blue4",
            col="blue4", col.main="blue4",col.lab="blue4",pch="+",
            cex=.45, cex.lab=1.2, cex.axis=1, cex.main=1.2)
plot(abd10,xvar=abdom$x,par=newpar)
```

Figure 12.4



R code on page 293

Figure 12.4: Residual plots from the BCT model abd10, where the `xvar` and `par` options have been modified

In order to see an application of the option (ts=TRUE) consider the aids data consisting of 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

qrt the quarterly seasonal effect a factor with 4 levels, [1=Q1 (Jan-March), 2=Q2 (Apr-June), 3=Q3 (July-Sept), 4=Q4 (Oct-Dec)]

Here we model the counts y using a negative binomial distribution with a (smooth) regression model in time x with a quarterly effect i.e. `cs(x,df=7)+qrt`, for the mean of y.

```
data(aids)
aids.1<-gamlss(y~cs(x,df=7)+qrt,family=NBI, data=aids)
```

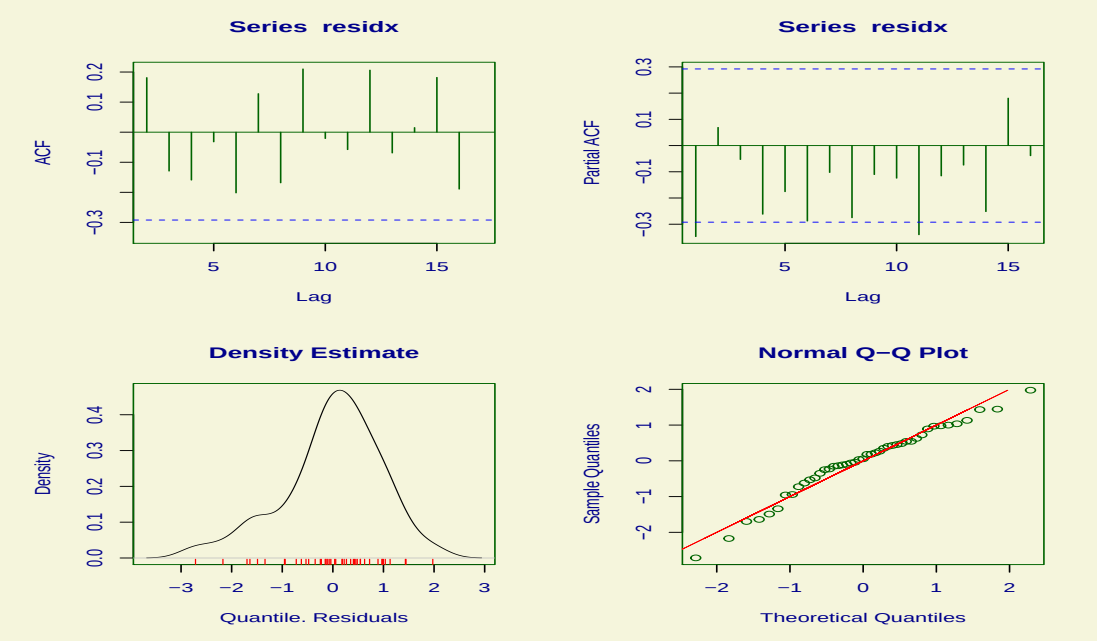
```
## GAMLSS-RS iteration 1: Global Deviance = 365.8129
## . . .
## GAMLSS-RS iteration 5: Global Deviance = 362.1123
```

The plot is shown in figure 12.5. It appears from the the ACF and the PACF functions shown in the top of figure 12.5 that the residuals do not show any systematic autocorrelation (since most of the values lie within the confidence intervals) therefore there is no further need to model the data using time series techniques.

Figure 12.5

```
plot(aids.1,ts=TRUE)
```

```
## ******************************************************************
##   Summary of the Randomised Quantile Residuals
##                            mean   =  -0.007775
##                        variance   =  0.9523
##                coef. of skewness  =  -0.6163
##                coef. of kurtosis  =  3.245
## Filliben correlation coefficient  =  0.9833
## ******************************************************************
```



R code on page 294

Figure 12.5: Residual plots from the NBI model fitted to the aids data

Note that since here we are using a discrete distribution family to model the data the residuals are randomised and the function `rqres.plot` should be used in addition to the function `plot`.

12.4 The wp() function

Worm plots of the residuals were introduced by van Buuren and Fredriks [2001] in order to identify regions (intervals) of an explanatory variable within which the model does not fit adequately the data (called "model violation"). The **R** function `wp` (which is based on the original `S-PLUS` function given in van Buuren and Fredriks [2001]) provides *single* or *multiple* worm plots for `gamlss` fitted objects. This is a diagnostic tool for checking the residuals for different ranges (by default not overlapping) of one or two explanatory variables. The worm plot is de-trended QQ- plots and the name comes from the worm like appearance of the plotted points.

single worm plot

If the `xvar` argument of the `wp()` function is not specified then a single worm plot is used. The following is an example of a single worm plot:

```
abd10<-gamlss(y~pb(x), sigma.fo=~pb(x), data=abdom, family=BCT)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 4771.925
## . . .
## GAMLSS-RS iteration 5: Global Deviance = 4770.993
```

```
wp(abd10)
```

Figure 12.6

The plot is shown in Figure 12.6. There are several important features in Figure 12.6:

- the (golden) points (or the worm) of the plot: These points show how far the residuals are from the their expected values represented in the figure by the horizontal dotted (red) line.
- the point-wise 95% confidence regions given by the two elliptic curves in the middle of the figure. If the model is correct we would expect approximately 95% of the points to be between the two elliptic curves and 5% outside. A higher percentage of the points outside the two elliptic curves indicates that the fitted distribution (or the fitted terms) of the model are inadequate to explain the response variable.
- the (red) fitted curve to the data: This curve is a cubic fit to the worm plot points. The shape of this cubic fit reflects different inadequacies in the model. Those are described in Table 12.1 and illustrated in Figure 12.7.

The important point here is that quadratic and cubic shapes in a worm plot indicate the presence of skewness and kurtosis respectively in the residuals. As far as Figure 12.7 is concerned since all the observations fall in the "acceptance" region inside the two elliptic curves and no specific shape is detected in the points, the overall model appears to fit well.

R code on page 295

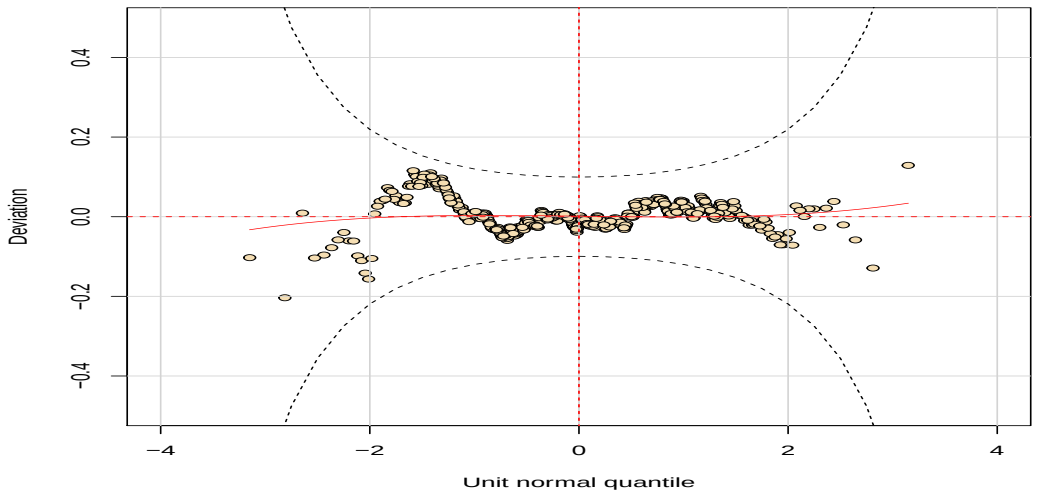


Figure 12.6: Worm plot from the BCT model `abd10` at default values

Table 12.1: The different shapes for the worm plot of the residuals (first column) and the corresponding deficiency in the residuals (second column) and deficiency in the response variable distribution (third column).

Shape of worm plot (or its fitted curve)	**Residuals**	**Response variable**
level: above the origin	mean too high	location parameter too low
level: below the origin	mean too low	location parameter too high
line: positive slope	variance too high	scale parameter too low
line: negative slope	variance too low	scale parameter too high
U-shape	positive skewness	skewness too low
inverted U-shape	negative skewness	skewness too high
S-shape with left bent down	lepto-kurtosis	kurtosis too low
S-shape with left bent up	platy-kurtosis	kurtosis too high

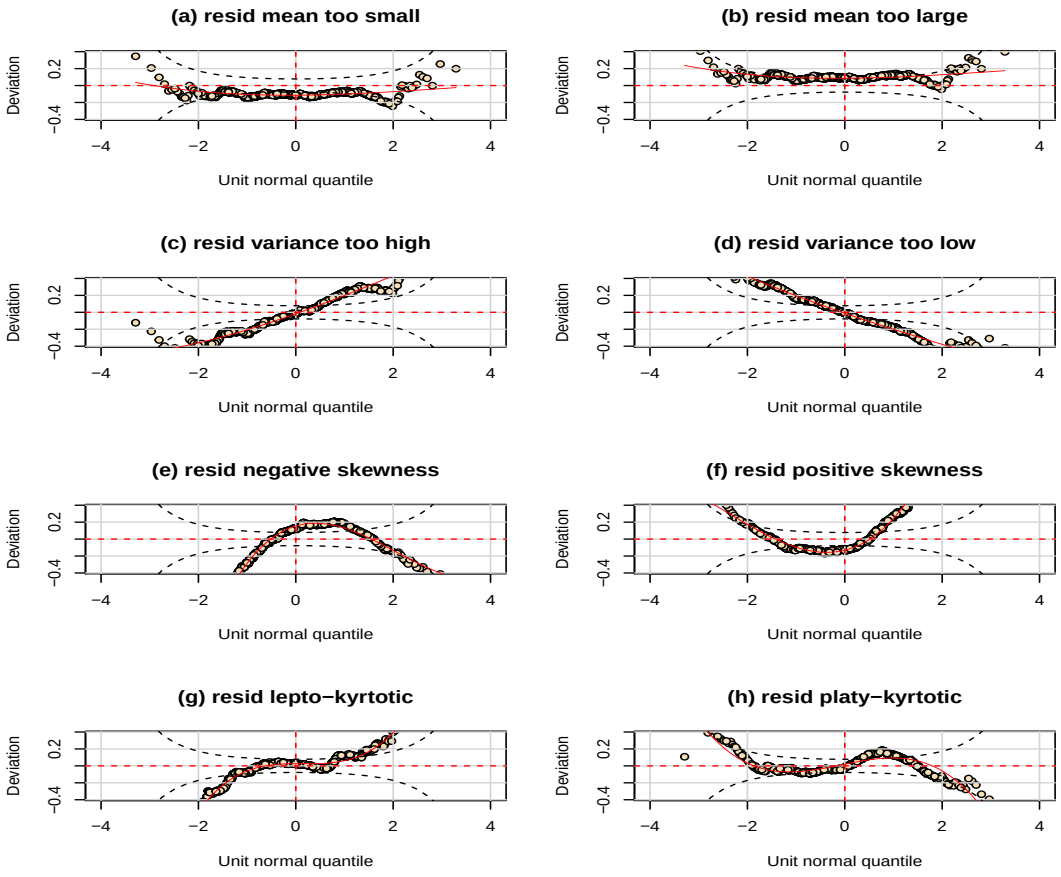


Figure 12.7: Different type of model failures indicated by the worm plot: i) plots (a) and (b) indicates failure for fitting correctly the location parameter with points falling below and above the horizontal (red) dotted line. ii) plots (c) and (d) indicates failure for fitting correctly the scale parameter. iii) plots (e) and (f) indicate failure for modelling the skewness in the data correctly and iv) plots (g) and (h) indicate failure for modelling the kurtosis

multiple worm plot

If the `xvar` argument of `wp()` is specified then we have as many worm plots as argument `n.iter` indicates. In this case the x-variable is cut into `n.iter` non-overlapping intervals with equal numbers of observations and the detrended normal QQ (i.e. worm) plots of the residuals for each interval are plotted. This is a way of highlighting failures of the model within different ranges of the explanatory variable. That is important when one of the explanatory variables is dominant in the analysis (as for example in centile estimation or in time series data). The parameters of the fitted cubic polynomials to the residuals in the worm plot can be obtained by e.g. `coRes <- wp(model1,xvar=x,n.iner=9)` and can be used as a way of checking the region in which the model does not fit adequately.

In the abdominal circumference example we are interested in whether the model fits well at the different regions of age. Here we are using the option `xvar` to specify age and `n.inter` to specify 9 intervals with equal number of observations for the worm plots. We are also saving the coefficient parameters of the fitted cubic polynomials for further diagnostics.

```
coef.1 <- wp(abd10,xvar=abdom$x,n.inter=9)
```

```
coef.1
```

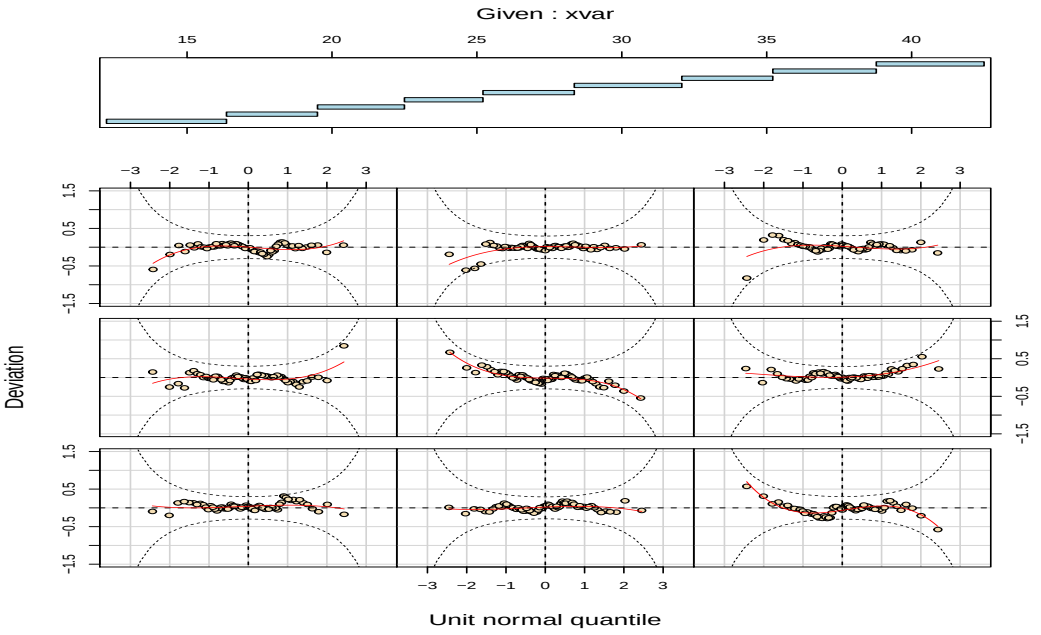
```
## $classes
##        [,1]  [,2]
##  [1,] 12.22 16.36
##  [2,] 16.36 19.50
##  [3,] 19.50 22.50
##  [4,] 22.50 25.21
##  [5,] 25.21 28.36
##  [6,] 28.36 32.07
##  [7,] 32.07 35.21
##  [8,] 35.21 38.78
##  [9,] 38.78 42.50
##
## $coef
##               [,1]         [,2]         [,3]         [,4]
##  [1,]  0.043594305  0.043925177 -0.005328564 -0.010167197
##  [2,]  0.020768938  0.042150024 -0.013714482 -0.009325333
##  [3,] -0.065344250  0.156093848  0.028205518 -0.067684581
##  [4,] -0.039820449 -0.059832136  0.028895496  0.029799686
##  [5,] -0.015744439 -0.013758616  0.011531208 -0.040762801
##  [6,]  0.009523141  0.027326782  0.045198638  0.006814800
##  [7,]  0.001338649 -0.077452581 -0.022366387  0.034188200
##  [8,]  0.014868877  0.005635431 -0.038020639  0.015777066
##  [9,]  0.024495114 -0.068089549 -0.020443334  0.022093903
```

Figure 12.8

```
wp(abd10,xvar=abdom$x,n.inter=9)
```

```
## number of missing points from plot= 0  out of  68
## number of missing points from plot= 0  out of  71
## number of missing points from plot= 0  out of  67
## number of missing points from plot= 0  out of  67
```

```
## number of missing points from plot= 0  out of  66
## number of missing points from plot= 0  out of  71
## number of missing points from plot= 0  out of  65
## number of missing points from plot= 0  out of  69
## number of missing points from plot= 0  out of  66
```



R code on page 298

Figure 12.8: Worm plot from the BCT model `abd10`

The resulting plot is shown in Figure **??** while table of intervals (`$classes`) above gives the 9 non-overlapping `x` (i.e. age) ranges in weeks. The worm plots are read from the bottom left corner, along each row in turn to the top right corner corresponding to the nine age intervals given in the (`$classes`) and plotted above the worm plots in steps.

The table of coefficients (`$coef`) gives in each column the fitted constant, linear, quadratic and cubic coefficients $\hat{\beta}_0$, $\hat{\beta}_1$, $\hat{\beta}_2$ and $\hat{\beta}_3$ respectively, for each of the nine cubic polynomials fitted to the nine detrended QQ-plots (for the nine non-overlapping rages of age given by `$classes`). van Buuren and Fredriks [2001] categorize absolute values of $\hat{\beta}_0$, $\hat{\beta}_1$, $\hat{\beta}_2$ and $\hat{\beta}_3$ in excess of threshold values 0.10, 0.10, 0.05 and 0.03 respectively, as misfits or model violations, indicating differences between the theoretical model residuals and the empirical mean, variance, skewness and kurtosis of the residuals respectively, within the particular age range (of the corresponding QQ-plot). Following these criteria in the above Table of coefficients, there are no misfits in $\hat{\beta}_1$, one misfit 0.15609 in $\hat{\beta}_2$, age group 3, no misfits in $\hat{\beta}_3$, and three misfits in $\hat{\beta}_4$ at 3rd, 5th and 7th range of age. [The number of misfits here may be due to the relative small sample size (610) for the abdominal data, relative to the sample sizes used by van Buuren and Fredriks [2001] leading to greater variance in the fitted parameters especially $\hat{\beta}_4$ than they experienced.]

The arguments of the `wp` function

For completeness we provide here all the arguments of the `wp()` function:

`object`	a `gamlss` fitted object or any other fitted model where the resid() method works (preferably it should produce quantile residuals)
`xvar`	the explanatory variable(s) against which the worm plots will be plotted. If only one variable is involved use `xvar=x1` if two variables are involved use `xvar=~x1*x2`. Factor can be used in the formula but not on their own, i.e. `xvar=~f1` is allowed but not `xvar=f1`.
`resid`	if object is missing this argument can be used to specify the residual vector (again it should be quantile residuals or it be assumed to come from a standard normal distribution)
`n.inter`	the number of intervals in which the explanatory variable `xvar` will be cut
`xcut.points`	the x-axis cut-off points e.g. `c(20,30)`. If `xcut.points=NULL` then the `n.inter` argument is activated
`overlap`	how much overlapping in the `xvar` intervals. Default value is `overlap=0` for non overlapping intervals
`xlim.all`	for a single worm plot this value is the x-variable limit, default is `xlim.all=4`
`xlim.worm`	for multiple worm plots this value is the x-variable limit, default is `xlim.worm=3.5`
`show.given`	whether to show the x-variable intervals in the top of the graph, default is `show.given=TRUE`
`line`	whether to plot the fitted cubic polynomial curve in each worm plot, default value is `line=TRUE`
`ylim.all`	for a single plot this value is the y-variable limit, default value is `ylim.all=12*sqrt(1/length(fitted(object)))`
`ylim.worm`	for multiple plots this value is the y-variable limit, default value is `ylim.worm=12*sqrt(n.inter/length(fitted(object)))`
`cex`	the cex plotting parameter with default `cex=1`
`pch`	the pch plotting parameter with default `pch=21`

12.5 the `Q.stats()` function

This function calculates and prints the Q-statistics which are useful to test normality of the residuals within ranges of an independent x-variable, for example age in centile estimation, see Royston and Wright [2000].

In order to explain what is a Q-statistic let us consider the situation where `age` is our main explanatory variable. Let G be the number of age groups and let $\{r_{gi}, i = 1, 2, .., n_i\}$ be the residuals in age group g, with mean $\bar{r}_g$ and standard deviation s_g, for $g = 1, 2, .., G$. The following statistics $\mathsf{Z}_{g1}, \mathsf{Z}_{g2}, \mathsf{Z}_{g3}, \mathsf{Z}_{g4}$ are calculated from the residuals in group g to test whether

the residuals in group g have population mean 0, variance 1, skewness 0 and kurtosis 3, (the values of standard normal distribution of the residuals assuming the model is correct), where

$$\mathsf{Z}_{g1} = n_g^{1/2}\bar{r}_g$$

,

$$\mathsf{Z}_{g2} = \left\{ s_g^{2/3} - [1 - 2/(9n_g - 9)] \right\} / \left\{ 2/(9n_g - 9) \right\}^{1/2}$$

and Z_{g3} and Z_{g4} are test statistics for skewness and kurtosis given by D'Agostino *et al.* (1990), in their equations (13) and (19) respectively. The Agostino $k2$ statistic, given by $k2_q = Z_{g3}^2 + Z_{g4}^2$, is a statistic for jointly testing whether he skewness of the residuals is different from 0 and the kurtosis is different from 3.

The Q statistics of Royston and Wright [2000] are then calculated by

$$Q_j = \sum_{g=1}^{G} \mathsf{Z}_{gj}^2$$

for $j = 1, 2, 3, 4$. Royston and Wright discuss approximate distributions for the Q statistics under the null hypothesis that the true residuals are normally distributed (although their simulation study was mainly for normal error models) and suggest Chi-squared distributions with adjusted degrees of freedom $G - df_\mu$, $G - [df_\sigma + 1]/2$ and $G - df_\nu$ for Q_1, Q_2 and Q_3 respectively. By analogy we suggest degrees of freedom $G - df_\tau$ for Q_4. The resulting significance levels should be regarded as providing a guide to model inadequacy, rather than exact formal test results.

Significant Q_1, Q_2, Q_3 or Q_4 statistics indicate possible inadequacies in the models for parameters μ, σ, ν and τ respectively, which may be overcome by increasing the degrees of freedom in the model for the particular parameter.

The Z_{gj} statistic when squared provides the contribution from age group g to the statistic Q_j, and hence helps identify which age groups are causing the Q_j statistic to be significant and therefore in which age groups the model is unacceptable.

Provided the number of groups G is sufficiently large relative to the degrees of freedom adjustment for the parameter, then the Z_{gj} values should have approximately standard normal distributions under the null hypothesis that the true residuals are standard normally distributed. We suggest as a rough guide values of $|\mathsf{Z}_{gj}|$ greater than 2 be considered as indicative of significant inadequacies in the model. Note that significant positive (or negative) values $\mathsf{Z}_{gj} > 2$ (or $\mathsf{Z}_{gj} < 2$) for $g = 1, 2, 3$ or 4 indicate respectively that the residuals have a higher (or lower) mean, variance, skewness or kurtosis than the null standard normal distribution. The model for parameter μ, σ, ν or τ may need more degrees of freedom to overcome this. For example if the residual mean in an age group is too high, the model for μ may need more degrees of freedom in order for the fitted μ from the model to increase within the age group. Note the Agostino $k2$ statistic $k2_q$ should be compared to the 5% value of a chi-square distribution with 2 degrees of freedom i.e. 6.0.

The following output is produced using the function `Q.stats` in the `abd10` model fitted in the previous section.

example

The following output is produced using the function `Q.stats` in the `abd10` model fitted in the previous Section.

Figure 12.9

```
qstats<-Q.stats(abd10,xvar=abdom$x,n.inter=9)
print(qstats, digits=3)
```

```
##                         Z1      Z2      Z3      Z4 AgostinoK2   N
## 12.22 to 16.36     0.3164  0.1858 -0.0621 -0.2573     0.0701  68
## 16.36 to 19.50     0.0615  0.1837 -0.2403 -0.3440     0.1761  71
## 19.50 to 22.50    -0.3083 -0.2341  0.4754 -2.4603     6.2789  67
## 22.50 to 25.21    -0.0939  0.2972  0.7637  1.2636     2.1798  67
## 25.21 to 28.36    -0.0360 -1.4162  0.2148 -1.6872     2.8928  66
## 28.36 to 32.07     0.4543  0.5490  0.8652  0.3267     0.8553  71
## 32.07 to 35.21    -0.1660  0.1988 -0.5205  0.9768     1.2250  65
## 35.21 to 38.78    -0.1865  0.5799 -0.8696  0.6295     1.1525  69
## 38.78 to 42.50     0.0361 -0.0294 -0.7508  1.2268     2.0687  66
## TOTAL Q stats      0.4791  2.8952  3.2564 13.6428    16.8992 610
## df for Q stats     3.1434  6.5475  8.0000  8.0000    16.0000   0
## p-val for Q stats  0.9346  0.8654  0.9173  0.0916     0.3922   0
```

The resulting plot of the Z-statistics is shown in Figure 12.9 where a misfit in the kurtosis statistic `Z4` at the range 22.5 to 25.21 is easily identified, as it is in the `Q.stats()` output. However in a table of 36 Z-statistics we would expect 2 to be significant at the 5% level by chance.

The original `Q.stats()` function was design for checking centile curve fitting, where a large number of data points are expected. The current version is more flexible allowing the input of residuals for models other than GAMLSS (suitable standardised) and also for smaller data sets. This happens with the use of the argument `resid` rather than `obj`. Here is an example of using `Q.stats()` with small data set of `aids`.

Figure 12.10

```
a1<-gamlss(y~pb(x)+qrt, family=PO, data=aids, trace=FALSE)
Q.stats(resid=resid(a1), xvar=aids$x, n.inter=5)
```

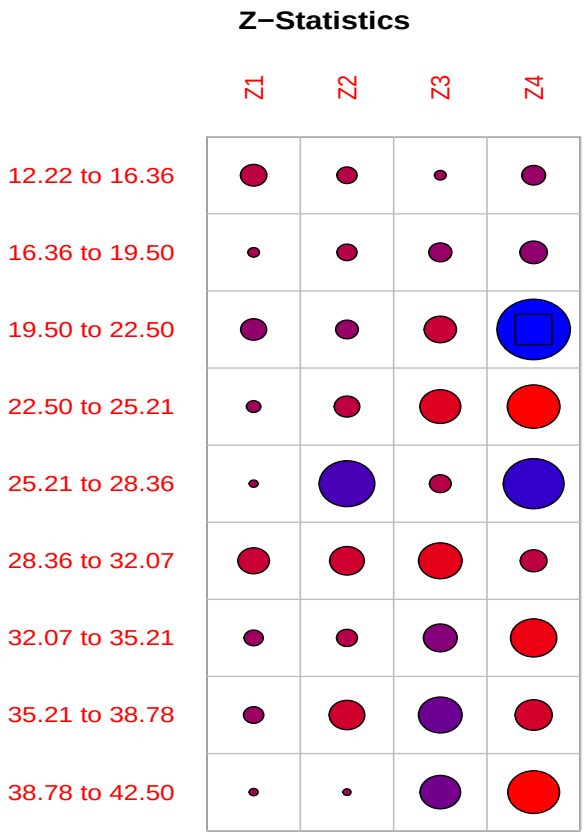
```
##                    Z1      Z2      Z3      Z4
##  0.5 to  9.5  0.26450 0.06712 -0.3188 -0.3402
##  9.5 to 18.5 -0.78511 0.30456 -0.4153  0.4598
## 18.5 to 27.5 -0.03934 0.83266  0.4262 -0.3823
## 27.5 to 36.5  0.22068 4.53123 -1.1096 -0.1319
## 36.5 to 45.5 -0.29971 0.74274 -1.4141  0.7248
```

The graphical presentation of the Z-statistics is shown in figure 12.10 where a misfit in the standard deviation in the interval 27.5 to 36.5 can be identified.

the arguments of the `Q.stats` function

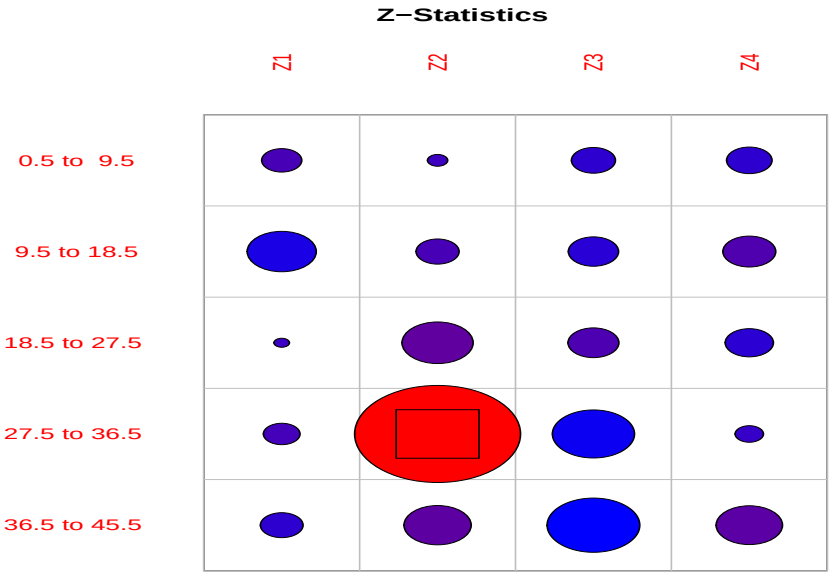
The `Q.stats` function has the following arguments

`obj` a `gamlss` object



R code on page 302

Figure 12.9: A visual presentation of the the Z statistics for the `abdom` model for easy identification of misfits in the data



R code on page 302

Figure 12.10: A visual presentation of the Z statistics for the `aids` model

`xvar`	the explanatory variable against which the Q statistics will be calculated
`resid`	quantile or standardised residuals can be given here instead of an `gamlss` object in `obj`. Note that the function `Q.stats` behaves differently depending whether the `obj` or the `resid` argument is set. The `obj` argument produces the Q-statistics (or Z-statistics) table appropriate for centile estimation (therefore it expect a reasonable large number of observations). The argument `resid` allows any model residuals, (not necessary GAMLSS), suitable standardised and is appropriate for any size of data. The resulting table contains only the individuals Z-statistics.
`xcut.points`	the x-axis cut off points e.g. `c(20,30)`. If `xcut.points=NULL` then the `n.inter` argument is activated
`n.inter`	the number of intervals in which the explanatory variable `xvar` will be cut
`zvals`	if `TRUE` the output matrix contains the individual Z statistics rather than the Q statistics
`save`	whether to save the Q (or Z) statistics or not with default equal to `TRUE`. In this case the functions produce a matrix giving individual Q (or Z) statistics and the final aggregate Q's
`plot`	whether to plot a visual version of the Q statistics (default is TRUE)

12.6 the rqres.plot() function

The function `rqres.plot()` is used to create different realisations of the normalised randomised quantile residuals [defined in Section 12.2] when the distribution of the response variable is discrete and plot then using worm plots or QQ-plots. Since randomisation is involved in discrete distributions the function `rqres.plot()` helps visually to decide whether the chosen distribution (and fitted terms) are an adequate represention of the data of not.

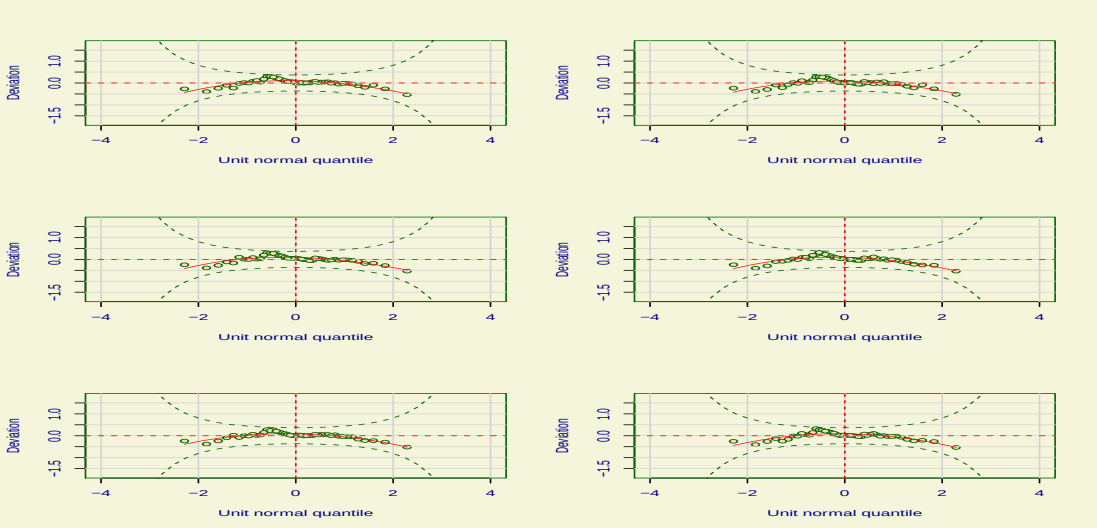
example

As an example we used the function `rqres.plot()` to plot residuals from a fitted model using the AIDS data:

```
m1 <- gamlss(y~pb(x)+qrt, data=aids, family=NBI, trace=FALSE)
```

```
rqres.plot(m1)
```

Figure 12.11



R code on page 305

Figure 12.11: Residual plots from the NBI model fitted to the aids data

The resulting plot is shown in figure **??**. Figure **??** shows six realisation of the worm plots from the randomised quantile residuals from the fitted model `m1` and in all six occasions the worms plots shown reasonable behaviour. For using QQ-plots instead of worm plots use `rqres.plot(m1, type="QQ")`.

We now try 40 realisation of the residuals and plot a QQ-plot of the mean of these realisations. The plot is shown in Figure **??**. Again the residuals appears to be reasonable. Hence the models seams adequate.

```
rqres.plot(m1, howmany=40,type="QQ",plot="average")
```

Figure 12.12

R code on page 305

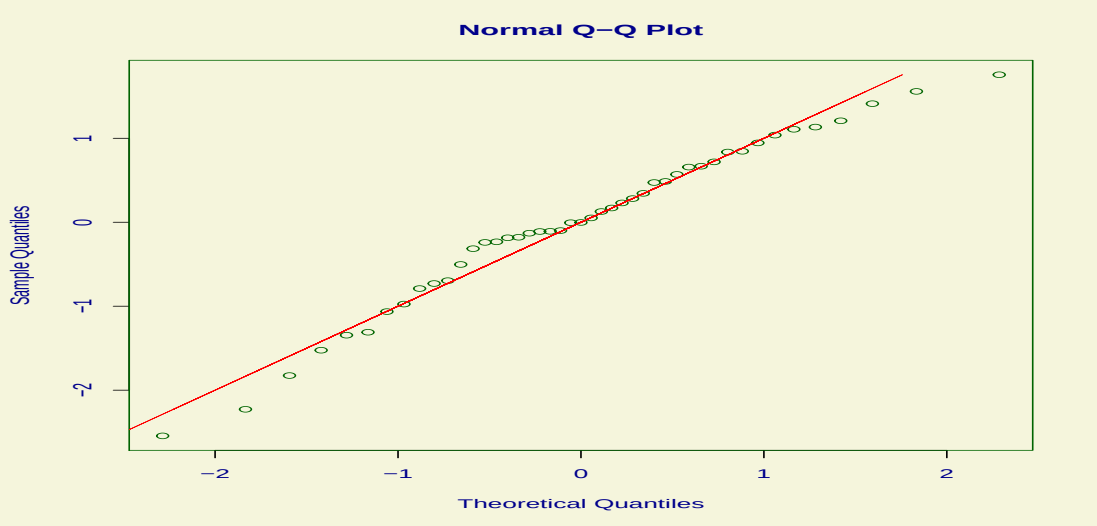


Figure 12.12: Residual plots from the NBI model fitted to the aids data

The arguments of the `rqres.plot()` function

It takes the following arguments.

`obj`	an `gamss` fitted model object from a discrete family
`howmany`	the number of worm or QQ-plots required up to ten, with default `howmany=6`
`plot`	whether to plot all plots, i.e. all the residual realisations `"all"` or just the mean `"average"`
`type`	whether to plot worm plots `"wp"`or QQ plots `"QQ"` with default worm plots

Part VI

Applications

Chapter 13

Centile Estimation

This chapter explains how to create growth curves using `gamlss`. In particularly it explains:

1. The LMS methods of centile estimation
2. The different functions for centile estimation within `gamlss`
3. How to use the functions effectively

This chapter is important for practitioners involved in centile estimation since GAMLSS has become one of the standard tools for creating centile growth curves.

13.1 Introduction

Centile estimation includes methods for estimating the age-related distribution of human growth. The standard estimation of centile curves usually involves two continuous variables:

1. the *response* variable, that is, the variable we are interested in and for which we are trying to find the centile curves, e.g. weight, BMI, head circumference etc.
2. the explanatory variable *age*.

The 100p centile of a random variable Y is the value y_p such that $p(Y \leq y_p) = p$, i.e. $y_p = F_Y^{-1}(p)$ so y_p is the inverse cumulative distribution function of Y (the `q` function in **R**) applied to p.

In this Chapter we consider the conditional centile of Y given explanatory variable $X = x$, i.e. $y_p(x) = F_{Y|X=x}^{-1}(p)$. By varying x a 100p centile curve of $y_p(x)$ against x is obtained. Centile curves can be obtained for different values of p. The World Health Organization uses the values 100p=(3,15,50,85,97) in its charts and 100p=(1,3,5,15,25,50,75,85,95,97,99) in its tables, see WHO [2006, 2007, 2009].

Centile estimation can be extended to more than one explanatory continuous variables e.g. age and height, see for example Cole et al. [2009] and Quanjer et al. [2012]. For categorical variables like gender the usual practice is to produce two separate charts against age.

Note that a z-score given the values of y and x is defined by $z_p = \Phi^{-1}\left[F_{Y|X=x}(y)\right]$, where Φ^{-1} is the inverse cumulative distribution function of a standard normal variable. For the values of y and x used in the estimation of the model the z-scores are the residuals of a fitted GAMLSS models see the definition of quantile residuals in Section 12.2 of Chapter 12.

The creation of sensible centile curves against age relies on non-parametric smoothing methods since parametric methods, e.g. polynomials or even fractional polynomials (Royston and Altman [1994]), are not in general flexible enough to encapsulate the features of the growth curve data. In smoothing methods the amount of smoothing depends on smoothing parameters and varies from data to data. The determination of the smoothing parameters is a crucial component of centile estimation. In the past several methods have been suggested which can be classified as:

1. Subjective (but structured) methods: The statistician (or practitioner) in this case uses his prior knowledge and experience in conjunction with some broad guidelines to choose the smoothing parameters and create the centile curves. For example first obtain a good smooth model for the location parameter then for the scale parameter and finally for the shape parameter(s) is one possible structured method. Erratic centile curves may indicate the need to increase the smoothing parameters.
2. Automatic methods: In a automatic procedure a criterion like for example the Akaike information criterion (AIC), or generalizations of it, can be used to select the smoothing parameters, Akaike [1973].
3. Methods based on diagnostics: In this case diagnostic tools like the worm plots of van Buuren and Fredriks [2001] or Royston and Wright [2000] can be used to determine the amount of smoothing. Poor worm plots or Q statistics may indicate the need to decrease the smoothing parameters, see for example Rigby and Stasinopoulos [2006b].

In reality a combination of all those procedures is a good practice.

The methodology for creating growth centile references for individuals from a population comprises two different methods:

i) the non parametric method of quantile regression (Koenker [2005]; Koenker and Bassett [1978], Koenker and Ng [2005] , He and Ng [1999] and Np and M. [2007])

ii) the parametric LMS (i.e. Lambda, Mu and Sigma) method of Cole [1988], Cole and Green [1992] and its extensions for example see Wright and Royston [1997], van Buuren and Fredriks [2001], and Rigby and Stasinopoulos [2004, 2006a].

In the next two section we describe the two approaches.

13.2 Quantile regression

Standard quantile regression methods estimate each quantile (i.e. centile) separately. He and Ng [1999] and Np and M. [2007] use smooth quantile curves using B-splines with a smoothness penalty. They developed the **COBS** and **quantreg** packages in `R`, respectively.

The following are features associated with quantile regression modelling:

- The quantile regression model does not assume a distribution for the response variable, therefore it is flexible and also in general reduces the bias caused by assuming a (possibly

wrong) distribution. This of course comes with a possible increase in the variability of the quantile curves (the usual bias against variance balance).

- The quantile curves near the extremes vary more than the ones in centre of the distribution of y and this is due to fact that those curves are supported by less observations. van Buuren [2007] commented that "curves produced by the quantile model are irregular near the extremes, and are generally less aesthetically pleasing" than the ones produced by parametric methods. This is more obvious by using the **(**COBS) and **quantreg** packages since each quantile curve is fitted separately. Quantile sheets, Schnabel and Eilers [2013a,b], do not suffer from this problem, since the estimation of the quantiles is done simultaneously. A function to fit quantile sheets, `quantSheets()`, is available in `gamlss` and it will be demonstrated in Section 13.11.

- A possible problem with quantile regression is that different quantile curves $y_p(x)$ for different values of p may cross (implying negative probability). There are several papers using quantile regression as a method to fit centile curves jointly, in order to overcome the problem, see for example Gannoun, A., Girard, S., Cuinot, C., and J [2002]; He [1997]; Heagerty and Pepe [1999]; LR and EJ [2005]; Wei et al. [2006]). However they result in restrictions on the quantile curves reducing their flexibility and therefore possibly increasing their bias.

- The quantile regression model does not allow for interpolation between quantile curves (for different p's) nor extrapolations beyond the outer centile curves which is desirable for tracking children with extreme growth.

- The fitted quantile regression model do not have a overall measure of fit, like GAIC, and this creates difficulties comparing competitive models.

- It is difficult to define the residuals of a fitted quantile regression model. Within `gamlss` and for a fitted `quantSheets` object this is achieved using an approximation. This approximation involves the function `flexDist()` which allows the user to reconstruct a distribution given the quantiles (and/or the expectiles).

- The fitted quantile regression model lacks an explicit formula allowing the calculation of quantile $y_p(x)$ given p and x, or the z-score given y and x. This was one of the requirements set by a World Health Organisation expert committee (Borghi et al. [2006]) for the adoption of a method for the construction of the world standard curves. This problem is related to the previous one and it is solved within the `gamlss` package using the function `flexDist()`.

13.3 The LMS method and extensions

The LMS method was developed by Cole [1988] and Cole and Green [1992] for fitting a single explanatory variable (age) to a response variable in order to create centile curves. Because the LMS method assumes that the y variable has a specific distribution, centile (quantile) curves for all p can be obtained and do not cross each other. Calculation of the quantile $y_p(x)$, given p and x, or the z-score, given y and x, are available for the LMS models.

The LMS method can be fitted within the `gamlss()` by assuming that the response variable has a Box-Cox Cole and Green distribution (BCCG) . The BCCG distribution is suitable for

positively or negatively skew data with $Y > 0$ and it is defined as follows:

Let the positive random variable $Y > 0$ be defined through the transformed random variable $\mathcal{Z}$ given by

$$\begin{aligned} \mathcal{Z} &= \frac{1}{\sigma\nu}\left[\left(\frac{Y}{\mu}\right)^{\nu} - 1\right], && \text{if} \quad \nu \neq 0 \\ &= \frac{1}{\sigma}\log\left(\frac{Y}{\mu}\right), && \text{if} \quad \nu = 0 \end{aligned} \tag{13.1}$$

for $0 < Y < \infty$, where $\mu > 0$, $\sigma > 0$ and $-\infty < \nu < \infty$, and where the random variable $\mathcal{Z}$ is assumed to follow a (truncated) standard normal distribution. The condition $0 < Y < \infty$ (required for Y^{ν} to be real for all ν) leads to the condition $-1/(\sigma\nu) < \mathcal{Z} < \infty$ if $\nu > 0$ and $-\infty < \mathcal{Z} < -\infty/(\sigma\nu)$ if $\nu < 0$, which necessitates the truncated standard normal distribution for $\mathcal{Z}$.

Rigby and Stasinopoulos [2004, 2006a] extended the LMS method (which models for skewness and but not for kurtosis in the data), by introducing the Box-Cox power exponential (BCPE) and the Box-Cox t (BCT) distributions and called the resulting methods LMSP and LMST respectively. The BCPE assumes that the transformed random variable $\mathcal{Z}$ has a (truncated) exponential power distribution, while BCT assumes that $\mathcal{Z}$ has a (truncated) t distribution. All these models are part of the GAMLSS framework, Rigby and Stasinopoulos [2005].

In the case of centile estimation for Y given an explanatory variable, e.g. *age*, the GAMLSS model is

$$\begin{aligned} Y &\sim D(\mu, \sigma, \nu, \tau) \\ g_1(\mu) &= h_1(x) \\ g_2(\sigma) &= h_2(x) \\ g_3(\nu) &= h_3(x) \\ g_4(\mu) &= h_4(x) \\ x &= age^{\xi} \end{aligned} \tag{13.2}$$

where the distribution D typically represents the BCCG, BCPE or BCT distributions, for which μ, σ, ν, and τ represent:

- the median,
- approximate coefficient of variation,
- skewness and
- kurtosis

parameters of the distribution respectively. Note that BCCG does not have τ. The $g()$ functions represent appropriate link functions, the $h()$ are non-parametric smoothing functions and ξ is a power transformation of age.

The power transformation, for age , ξ, is usually needed when the response variable has an early or late spell of fast growth. In those cases the transformation of age can stretch the time scale making the smooth curve fitting easier.

Each link function, $g()$, is usually chosen to ensure that the parameters are defined appropriately. For example a log() link function ensures that the parameter in question remains positive. Note

however, that the original formulation of the LMS method introduced by Cole and Green [1992] uses identity link for all the parameters of BCCG. Also for historical reason the first formulation of the BCCG, BCPE and BCT distributions has identity link function sfor μ as a default, even though μ should be always positive. The distributions BCCGo, BCPEo and BCTo all have a log link as a default for μ.

The non-parametric smoothing functions $h()$ usually require the specification of a smoothing parameter λ or the equivalent degrees of freedom to be used, see for example Hastie and Tibshirani [1990] and Wood [2006]. Next we describe the methods used within `gamlss`.

13.3.1 Model selection procedures for the LMS method

The selection of the the link functions $g_k(.)$, for $k = 1, 2, 3, 4$ usually does not create a problem. Log link functions are preferable for σ and τ (to ensure $\sigma > 0$ and $\tau > 0$). The identity link function is appropriate for ν since $-\infty < \nu < \infty$. For μ the safe option is to use the "log" link by using the BCCGo, BCPEo and BCTo distributions, but for most cases the identity link works (distributions BCCG, BCPE and BCT). [The preferred link function is the one for which the fitted model has the smaller value of $GAIC(K)$ for a particular penalty k (e.g. $k = 3$).]

Given the link functions, the model specification comprises now finding the (effective) degrees of freedom for the smooth non-parametric terms $h_k(x)$ for $k = 1, 2, 3, 4$, denoted df_μ, df_σ, df_ν and df_τ respectively, and ξ in the transformation for age, $x = age^\xi$. That is, we have to select the five 'hyperparameters' (df_μ, df_σ, df_ν,df_τ, and ξ).

Over the years different procedures have been considered by the authors. Here we explain three of procedures used for choosing the hyperparameters. Table 11.1 from Chapter **??** shows where information about the different methods can be obtained.

Method 1: This method is minimising the $GAIC(k)$ over the five hyperparameters (df_μ, df_σ, df_ν,df_τ,ξ). Rigby and Stasinopoulos [2006a] used as an automatic procedure, the function `find.hyper()` which is based on the numerical optimisation function `optim()` in **R**, to minimise the generalised Akaike information criterion $GAIC(k)$, over the five hyperparameters, the four total (effective) degrees of freedom df_μ, df_σ, df_ν, df_τ and the power transformation parameters ξ. They used the BCT distribution model (13.2) and different values of the penalty k including AIC ($k = 2$) and SBC ($k = \log(n)$). They have found that the value $k = 3$ was a good compromise between the two well known criteria and produced good looking growth curves.

Method 2: This method minimizes the Validation Global Deviance (VGD) over the five hyperparameters, Stasinopoulos and Rigby [2007]. In this procedure the data were split randomly into 60% training and 40% validation data sets. For each specific set of hyperparameters, model (13.2) was fitted to the training data and the resulting validation global deviance $VGD = -2\hat{l}_v$, where $\hat{l}_v$ is the log likelihood of the validation data given the fitted training data model (13.2), was calculated. VGD was then minimised over the five hyperparameters using the numerical optimisation function `optim()`.

Method 3: This method has two steps. In first step , if transformation on the x-axis is needed, them for then for the simple model $g(\mu) = s(x^\xi)$ the GAIC(k) is minimised over ξ . Given the estimated ξ, the second step involves the estimation of the four degrees of freedom hyperparameters (df_μ, df_σ, df_ν,df_τ) using a local ML procedure, Rigby and Stasinopoulos

[2013]. This is the three fastest method and results to nmodels with similar centiles to the two previous ones.

Next we are consider an example.

13.4 The Dutch boys BMI data

For the next sections of this Chapter we will use data from the Fourth Dutch Growth Study, Fredriks, A.M., van Buuren, S., Burgmeijer, R.J.F., Meulmeester, J.F., Beuker, R.J., Brugman, E., Roede, M.J., Verloove-Vanhorick, S.P. and Wit [2000], Fredriks, A.M., van Buuren, S., Wit, J.M. and Verloove-Vanhorick [2000] which is a cross-sectional study that measures growth and development of the Dutch population between the ages 0 and 21 years. The study measured, among other variables, height, weight, head circumference and age for 7482 males and 7018 females. The data were kindly provided by Professor Stef. van Buuren.

Here we have only the BMI, (y), and age, x, of Dutch boys as explanatory variable and we are interested also in a transformation of age $x = age^{\xi}$. Cases with missing values have been removed. There are 7040 observations. The data are plotted in Figure 13.1.

Figure 13.1

```
library(gamlss)
data(dbbmi)
plot(bmi~age, data=dbbmi, pch = 15, cex = 0.5, col = gray(0.5))
```

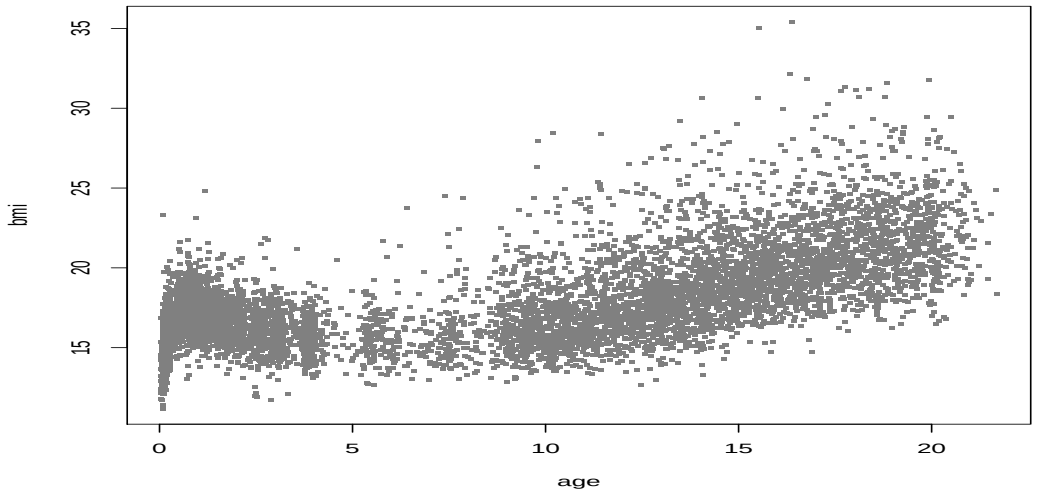


Figure 13.1: BMI against the age of the Dutch boys data

13.5 The lms() function

The function lms() is designed to facilitate the automatic selection of an appropriate LMS method model for the construction of growth curves. In particular i) the determination of the distribution of the response variable, ii) the appropriate degrees of freedom for all the parameters of the distribution and iii) the power parameter ξ. This avoid a global GAMLSS modelling selection. Note though that lms() is applicable with "one" explanatory variable only. The function lms() has the following arguments:

y	The response variable
x	The unique explanatory variable, usually age
families	a list of gamlss.families with default LMS=c("BCCGo", "BCPEo", "BCTo"). Note that this list is appropriate for positive response variables.
data	the data frame
k	the penalty to be used in the GAIC, with default value $k = 2$
cent	a vector with elements the % centile values for which the centile curves have to be evaluated
calibration	whether calibration is required with default TRUE, (see Section 13.7.2)
trans.x	whether to check for transformation in x with default FALSE
lim.trans	the limits for the search of the power parameter for x
legend	whether a legend is required in the plot with default FALSE
mu.df	mu effective degrees of freedom if required, otherwise it is estimated
sigma.df	sigma effective degrees of freedom if required, otherwise it is estimated
nu.df	nu effective degrees of freedom if required, otherwise it is estimated
tau.df	tau effective degrees of freedom if required, otherwise it is estimated
method.pb	the method used in the pb() for local estimation of the smoothing parameters. The default is local maximum likelihood "ML". "GAIC" is also permitted where
k	is taken from the k argument of the function.
...	extra arguments which can be passed to gamlss

An example of using the lms() function is given below. To show the usage of the functions we have taken a sample of 1000 observations from the original 7040 observations of dbbmi data for speed. The sample data are plotted in Figure 13.2.

```
set.seed(2803)
IND<-sample.int(7040, 1000, replace=FALSE)
dbbmi1 <- dbbmi[IND,]
plot(bmi~age, data=dbbmi1, pch = 15, cex = 0.5, col = gray(0.5))
```

Figure 13.2

The BMI data of Figure 13.1 and 13.2 show a fast growth in BMI for children during the first year from birth indicating that a power transformation for age could be appropriate for this data. Therefore the argument trans.x = TRUE if the function lms() is used.

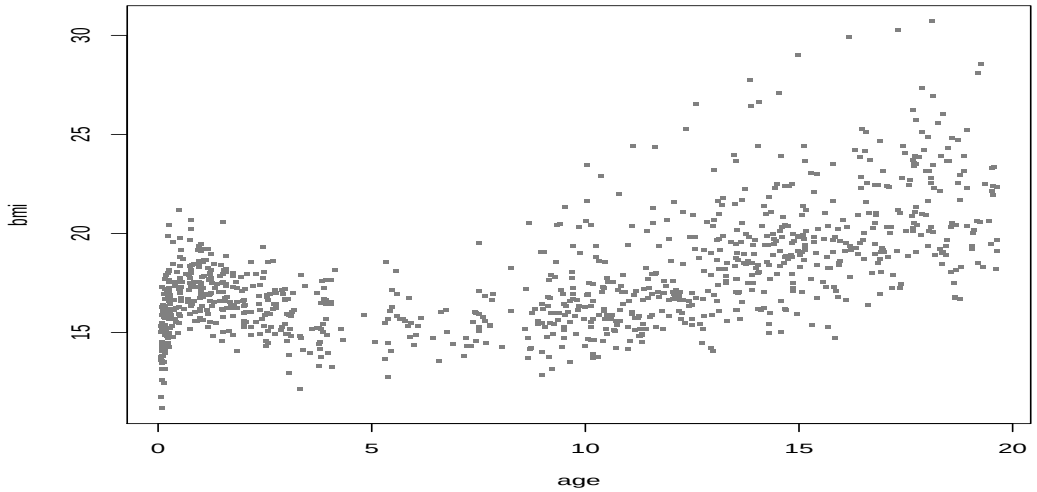


Figure 13.2: Sample of BMI against the age of the Dutch boys data

```
m0 <-  lms(bmi,age, data=dbbmi1, trans.x=T, k=2)
```

```
## *** Checking for transformation for x ***
## *** power parameters  0.02764176 ***
## *** Initial  fit***
## GAMLSS-RS iteration 1: Global Deviance = 4261.962
## . . .
## GAMLSS-RS iteration 9: Global Deviance = 3940.004
## GAMLSS-RS iteration 10: Global Deviance = 3940.005
## % of cases below  0.5229901 centile is  0.4
## % of cases below  1.89542 centile is  2.3
## % of cases below  8.695653 centile is  9.2
## % of cases below  25.72263 centile is  25.3
## % of cases below  50.09397 centile is  50
## % of cases below  74.93388 centile is  74.7
## % of cases below  90.72126 centile is  90.8
## % of cases below  97.86072 centile is  97.7
## % of cases below  99.40448 centile is  99.6
```

```
m0$family
```

```
## [1] "BCCGo"                    "Box-Cox-Cole-Green-orig."
```

The transformation chosen for age is $x = age^{0.028}$ and the best distribution according to GAIC was BCCGo. Note however that if all the 7294 observations were included in the fit using say

```
lms(bmi,age, data=dbbmi, trans.x=TRUE)
```

the power transformation parameter would have been $x = age^{0}.436$ and the final distribution BCTo not BCCGo.

Checking the fitted model using residual diagnostics is very important for the creation of growth curves. The worm plots, `wp()` and the Q-statistics, described in Chapter 12, are two of those methods :

Figure 13.3

```
round(Q.stats(m0, xvar=dbbmi1$age),3)
```

```
##                         Z1     Z2     Z3     Z4 AgostinoK2    N
##  0.055 to  0.265  -0.732 -0.298 -0.060  0.608      0.373   92
##  0.265 to  0.875  -0.322  0.224 -0.939 -0.687      1.354  152
##  0.875 to  1.625  -0.067  0.362 -1.199 -0.356      1.565  205
##  1.625 to  3.035   0.465 -0.564 -0.771  0.342      0.711  248
##  3.035 to  7.435   0.645 -0.117 -0.363  0.530      0.413  279
##  7.435 to  9.965   0.112 -0.568 -0.158  0.589      0.372  287
##  9.965 to 11.415   0.620  0.179  1.157  0.289      1.423  292
## 11.415 to 13.095   0.461  0.026  0.459  0.282      0.291  281
## 13.095 to 14.425   0.975 -0.514 -0.312  0.302      0.189  251
## 14.425 to 15.865   0.827 -0.587 -0.601 -0.046      0.363  208
## 15.865 to 17.715   0.708 -0.448  0.092 -0.747      0.567  157
## 17.715 to 19.645   0.063  0.302 -0.031 -1.143      1.308   83
## TOTAL Q stats      4.024  1.858  5.093  3.835      8.928 2535
## df for Q stats     2.020  9.430  6.148 12.000     18.148    0
## p-val for Q stats  0.136  0.996  0.550  0.986      0.964    0
```

The plot is given in Figure 13.3 indicates that the Q-statistics seems reasonable for all the parameters of the model.

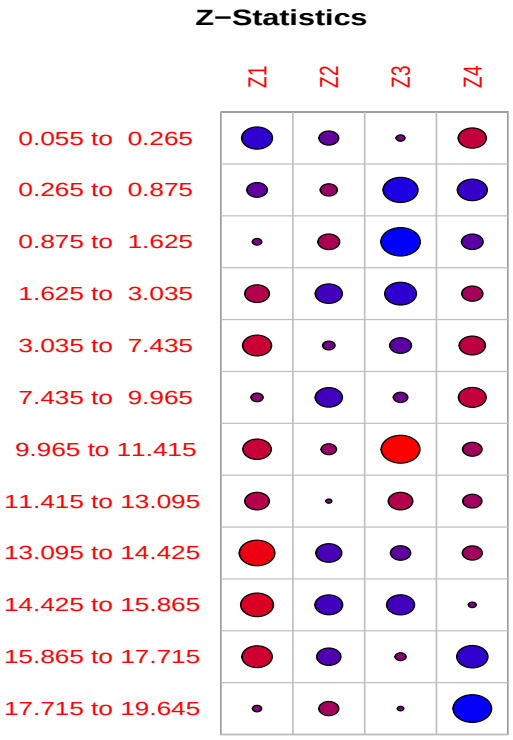
Now we are checking the worm plots.

Figure 13.4

```
wp(m0, xvar=dbbmi1$age, n.inter=9)
```

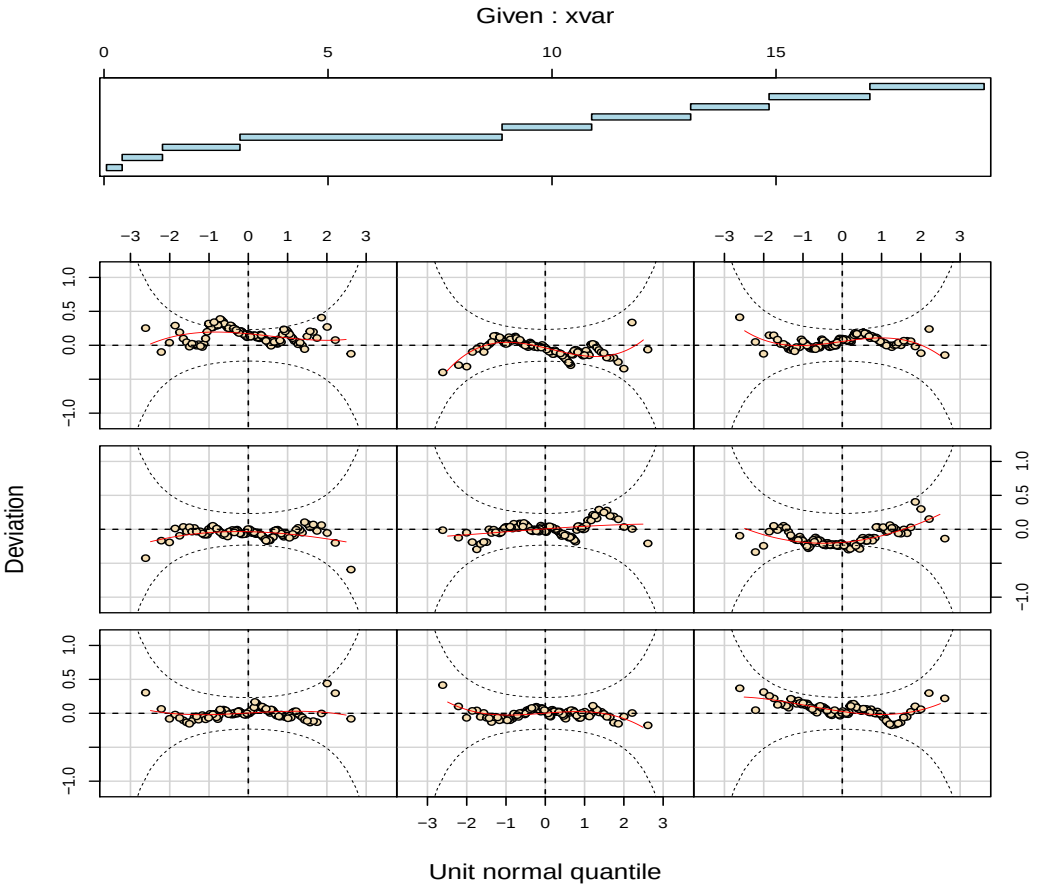
```
## number of missing points from plot= 0  out of  112
## number of missing points from plot= 0  out of  111
## number of missing points from plot= 0  out of  110
## number of missing points from plot= 0  out of  112
## number of missing points from plot= 0  out of  111
## number of missing points from plot= 0  out of  111
## number of missing points from plot= 0  out of  112
## number of missing points from plot= 0  out of  110
## number of missing points from plot= 0  out of  111
```

The plot given in Figure 13.4 shows that the residuals look good for all 9 intervals of age, indicating that the model is adequate.



R code on page 317

Figure 13.3: A plot of Q-statistics for the fitted `lms` object `m0`



R code on page 317

Figure 13.4: A worm plot for the fitted `lms` object `mo`

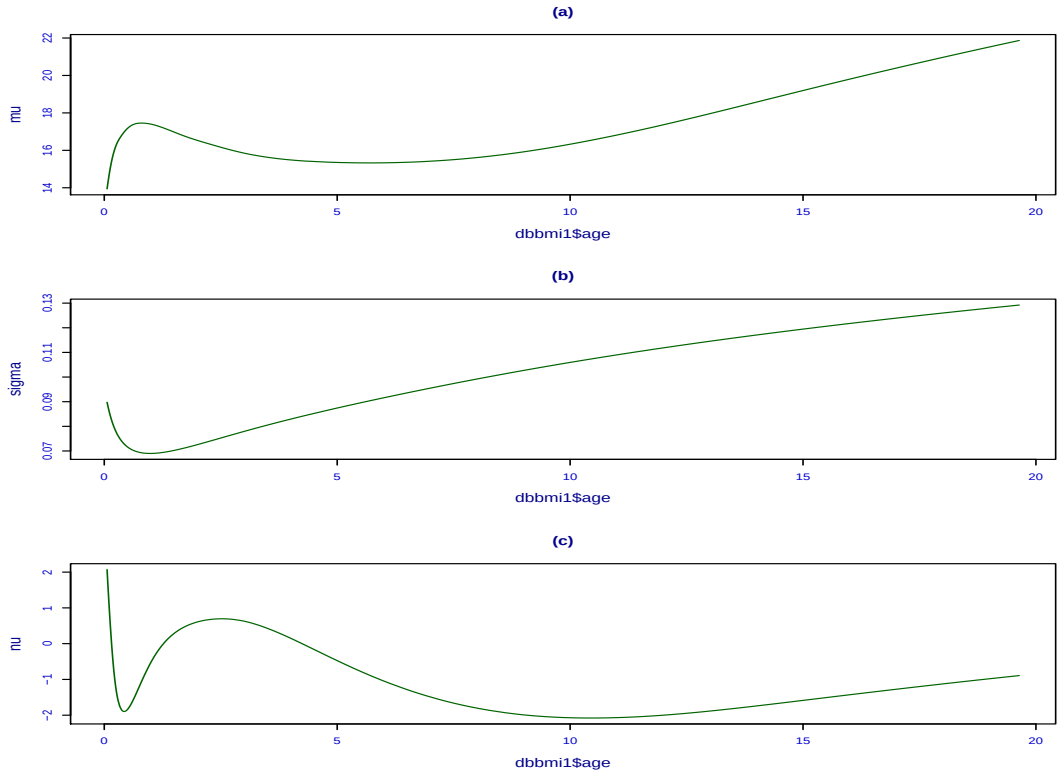
13.6 Plotting fitted values against the x variable using `fittedPlot()`

The function `fittedPlot()` provides a convenient way of plotting the fitted μ, σ, ν and τ if the fitted model involves only one explanatory variable say x. Therefore it can be used after the `lms()` function.

Figure 13.5

```
fittedPlot(m0 ,x=dbbmi1$age)
```

The plot is given in Figure 13.5.



R code on page 320

Figure 13.5: The fitted values for all four parameters against age, from a Box-Cox Colen Green (BCCGo) distribution fitted using the BMI data, i.e. fitted values of (a) μ (b) σ and (c) ν

The `fittedPlot()` function has the following arguments

`object`	a fitted `gamlss` model object (with only one explanatory variable)
`...`	optionally more fitted `gamlss` model objects
`x`	the unique explanatory variable

`color`	whether the fitted lines in the plot are shown in colour, 'color=TRUE' (the default) or not 'color=FALSE'
`line.type`	whether the line type should be different or not. The default is `color=FALSE`
`xlab`	the x-label

The fitted values of more that one model can also be plotted together using `fittedPlot`. For example here we compare model `m0` with model `m1` which is fitted using the BCPEo distribution and `pb()` with fixed smoothing degrees of freedom `df` for each parameter predictor.

```
Tage=(dbbmi1$age)^(m0$power)
m1 <- gamlss(bmi~pb(Tage), sigma.formula=~pb(Tage),
               nu.formula=~pb(Tage), tau.fomula=~pb(Tage), family=BCPEo,
               data=dbbmi1)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 3979.317
## . . .
## GAMLSS-RS iteration 7: Global Deviance = 3953.243
```

```
fittedPlot(m1,m0, x=dbbmi1$age, line.type=c(1,2))
```

Figure 13.6

The plot is given in figure 13.6. Note that the fitted values for τ for the BCPEo are flat indicating a constant model.

13.7 Plotting centiles curves using `centiles()` and `calibration()`

Centile plots are currently provided for all the continuous distributions in Table **??**.

There are three functions for plotting centiles i) the `centiles`, ii) `centiles.fan` and ii) the `centiles.split` which are described in sub-sections 13.7.1 and 13.8 respectively

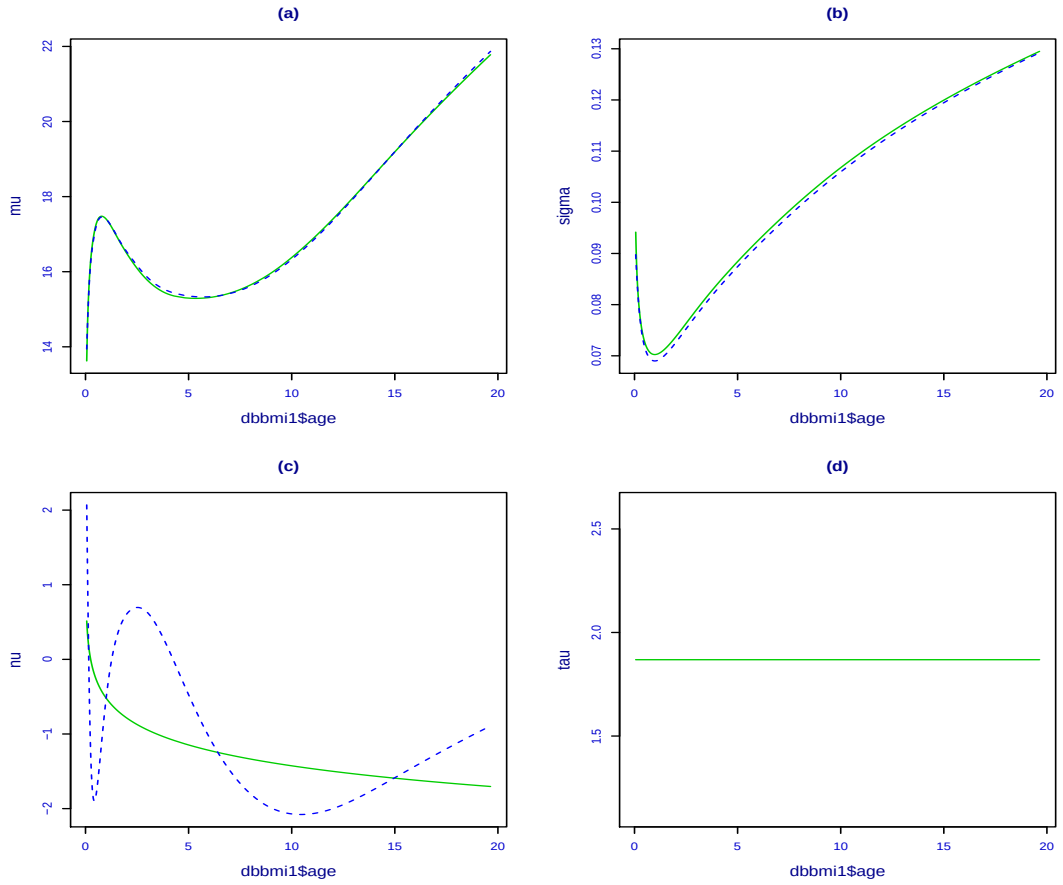
13.7.1 The function `centiles()`

For a simple use try `centiles()`, see Figure 13.7(a) for the plot. Note that the function `calibration()` automatically prints the sample percentage of observations below each of the fitted centiles from the fitted model, so comparisons with nominal model %'s can be made. In figure 13.7 (b) The sample % are close to the nominal model %'s.

Figure 13.7

```
op <- par(mfrow=c(2,1))
centiles(m0,dbbmi1$age, main="(a)", legend=FALSE)
```

```
## % of cases below  0.4 centile is  0.3
## % of cases below  2 centile is  2.5
## % of cases below  10 centile is  10.6
## % of cases below  25 centile is  24.5
## % of cases below  50 centile is  49.9
## % of cases below  75 centile is  74.8
## % of cases below  90 centile is  89.8
## % of cases below  98 centile is  97.8
```



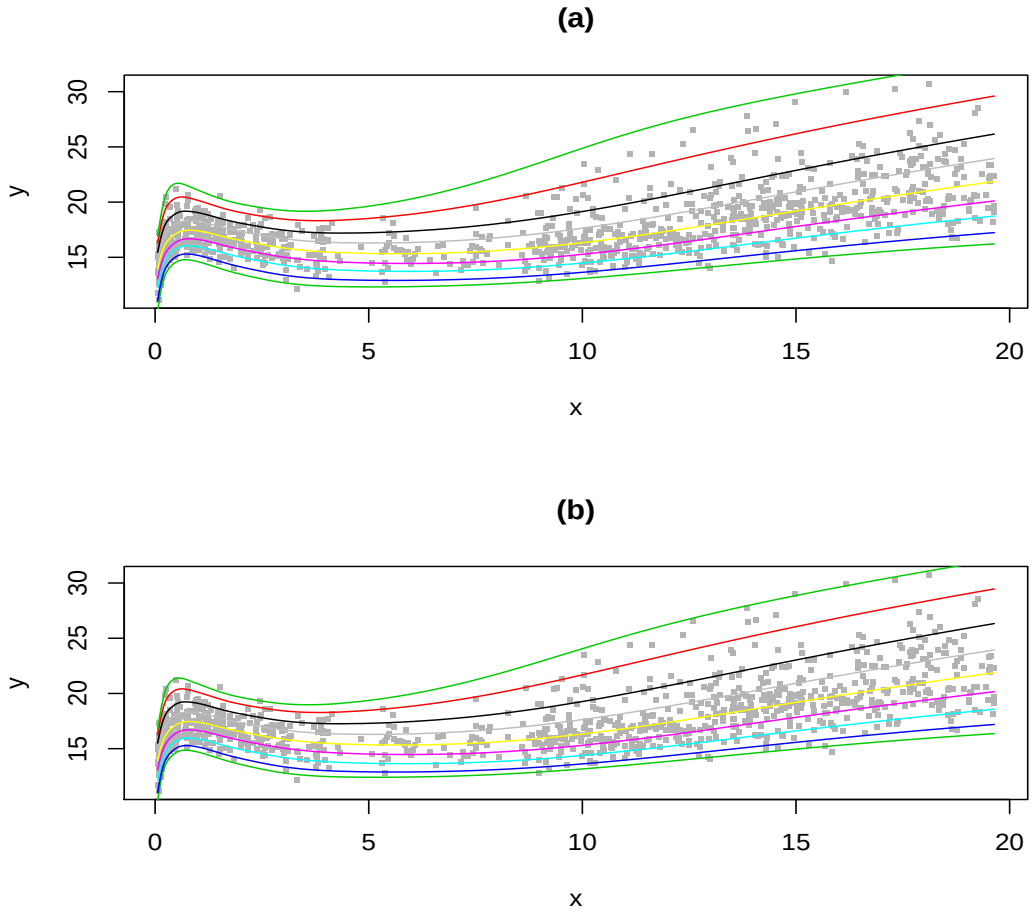
R code on page 321

Figure 13.6: Comparing the fitted values for all parameters against the transformed age, for models the BBCGo model `m0`, solid line, and the BCPEo model `m1`, dash line: (a) μ (b) σ (c) ν (d) τ

```
## % of cases below  99.6 centile is  99.9

calibration(m0,dbbmi1$age, main="(b)")

## % of cases below  0.5229901 centile is  0.4
## % of cases below  1.89542 centile is  2.3
## % of cases below  8.695653 centile is  9.2
## % of cases below  25.72263 centile is  25.3
## % of cases below  50.09397 centile is  50
## % of cases below  74.93388 centile is  74.7
## % of cases below  90.72126 centile is  90.8
## % of cases below  97.86072 centile is  97.7
## % of cases below  99.40448 centile is  99.6

par(op)
```

The following are the arguments of the function `centiles`

`obj`	a fitted `gamlss` object
`xvar`	the unique explanatory variable for which we would like the fitted model centiles to be calculated
`cent`	a vector with elements the % centile values for which the fitted model centile curves have to be evaluated. e.g. if you wish % centiles at points 5% and 95% only, use `cent= c(5, 95)`
`legend`	whether a legend is required within the plot or not, the default is `legend=TRUE`. This legend identifies the different centile curves and it is boxed.
`ylab`	the y-variable label
`xlab`	the x-variable label
`main`	the main title here as character. If NULL the default title "centile curves using NO" (or the relevant distributions name) is shown
`main.gsub`	if the `main.gsub` (with default "@") appears in the main title then it is substituted with the default title.
`xleg`	position of the legend in the x-axis
`yleg`	position of the legend in the y-axis
`xlim`	the limits of the x-axis
`ylim`	the limits of the y-axis
`save`	whether to save the sample percentages or not with default equal to 'FALSE'. In this case the sample percentages are printed but are not saved
`plot`	whether to plot the centiles. This option is useful for 'centile.split'
`pch`	the character to be used as the default in plotting points, see the option for `par()` .i.e. `?par`
`cex`	size of character, see `par`



R code on page 321

Figure 13.7: Centiles curves (a) and calibration curves (b) using Box-Cox Colen Green (BCCGo) distribution for the BMI data

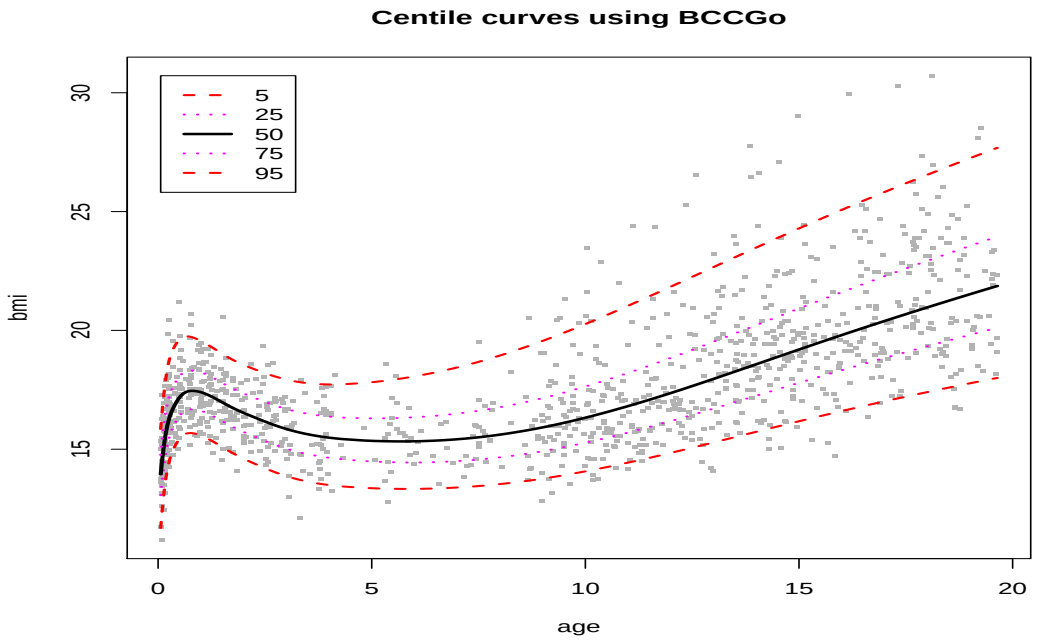
`col`	the colour of points, see `par`
`col.centiles`	the colours for the centile curves
`lty.centiles`	the line types for the centile curves
`lwd.centiles`	the line width for the centile curves
`points`	whether the data points should be plotted
`...`	for extra arguments

As an example, a modified version of the centiles in Figure **??** is given below. See figure 13.8 for the plot.

Figure 13.8

```
centiles(m0,dbbmi1$age,cent=c(5,25,50,75,95), ylab="bmi", xlab="age",
         col.centiles = c(2,6,1,6,2), lty.centiles = c(2,3,1,3,2),
         lwd.centiles =c(2,2,2.5,2,2))
```

```
## % of cases below  5 centile is  5
## % of cases below  25 centile is  24.5
## % of cases below  50 centile is  49.9
## % of cases below  75 centile is  74.8
## % of cases below  95 centile is  95
```



R code on page 325

Figure 13.8: Centile curves using Box-Cox t (BCT) distribution for the BMI data

Note that the output obtained from the `centiles()` function can be useful to get information on how well a distribution fits at a particular age. Here we selected the cases from the Dutch

boys data set at the rounded age of 10 (ie between 9.5 and 10.5), and fit a BCCGo distribution to the sample.

```
sub1<-subset(dbbmi, (age > 9.5 & age < 10.5))
```

```
h1 <- gamlssML(bmi, data=sub1, family=BCCGo)
```

Figure 13.9

```
centiles(h1,sub1$age,cent=c(1,2.5, 10, 25, 50, 75, 90, 97.5, .99), legend=FALSE)
```

```
## % of cases below  1 centile is  0.2777778
## % of cases below  2.5 centile is  3.611111
## % of cases below  10 centile is  8.888889
## % of cases below  25 centile is  26.11111
## % of cases below  50 centile is  50.83333
## % of cases below  75 centile is  77.5
## % of cases below  90 centile is  89.72222
## % of cases below  97.5 centile is  97.22222
## % of cases below  0.99 centile is  0.2777778
```

See Figure 13.9 for the plot. The fit did not capture well the 1% and the 99 % tails of the BMI.

R code on page 326

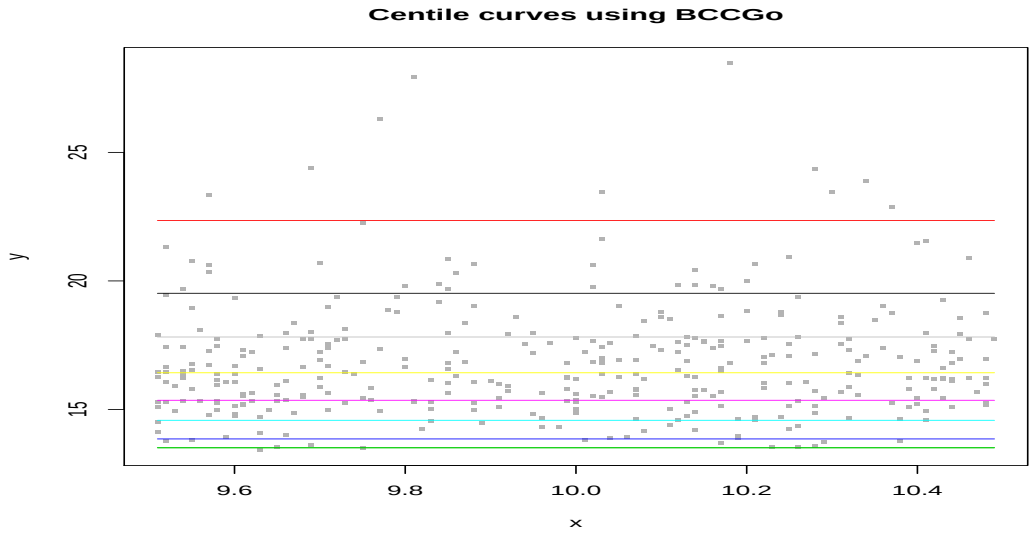


Figure 13.9: Centile curves using Box-Cox Cole and Green distribution to fit BMI at rounded aged 10 for the Dutch boys data

If no variable is available the user can create an index variable by `index<-1:n` , where n is the number of observations and use this in the centiles command, i.e. `centiles(h1,index)`. An alternative way to check the distribution at rounded age 10 is to use function `histDist()`, ie `histDist(y,family="NO", data=sub1)`.

13.7.2 The function calibration()

This function can be used when the fitted model centiles do not coincide with the sample centiles and it is assumed that this failure is the same for all values of the explanatory variable, xvar. The calibration function finds the sample quantiles of the residuals of the fitted model (the z-scores) and uses them as sample quantile in the argument cent of the centiles() function. Consider the following example of the calibration function.

```
calibration(m0,xvar=dbbmi1$age)
```

```
## % of cases below  0.5229901 centile is  0.4
## % of cases below  1.89542 centile is  2.3
## % of cases below  8.695653 centile is  9.2
## % of cases below  25.72263 centile is  25.3
## % of cases below  50.09397 centile is  50
## % of cases below  74.93388 centile is  74.7
## % of cases below  90.72126 centile is  90.8
## % of cases below  97.86072 centile is  97.7
## % of cases below  99.40448 centile is  99.6
```

See Figure 13.7(b) for the plot. In this case that calibration() function produce similar results with the function centiles(). The calibration() function apart from object, xvar and cent has as arguments:

legend	whether legend is required (default is FALSE).
fan	for fan plots (default is FALSE.

13.7.3 The function centiles.fan()

The function centiles.fan() plots a fan-chart of the centile curves.

```
centiles.fan(m0,dbbmi1$age,cent=c(5,25,50,75,95), ylab="bmi", xlab="age")
```

Figure 13.10

See Figure 13.10 for the plot. The different colour schemes to be used for the fan-chart are "cm","gray", "rainbow", "heat", "terrain" and "topo".

13.8 The function centiles.split()

The function centiles.split() splits the fitted centile curves according to different cut points in the x-variable (age). Here we split the centiles plot at $x = 2$ (e.g. xcut.points=c(2)):

Figure 13.11

```
centiles.split(m0,xvar=dbbmi1$age,xcut.points=c(2))
```

```
##      0.06 to 2 2 to 19.64
## 0.4      0.000     0.4098
## 2        2.985     2.3224
## 10      11.194    10.3825
## 25      23.507    24.8634
## 50      48.507    50.4098
```

R code on page 327

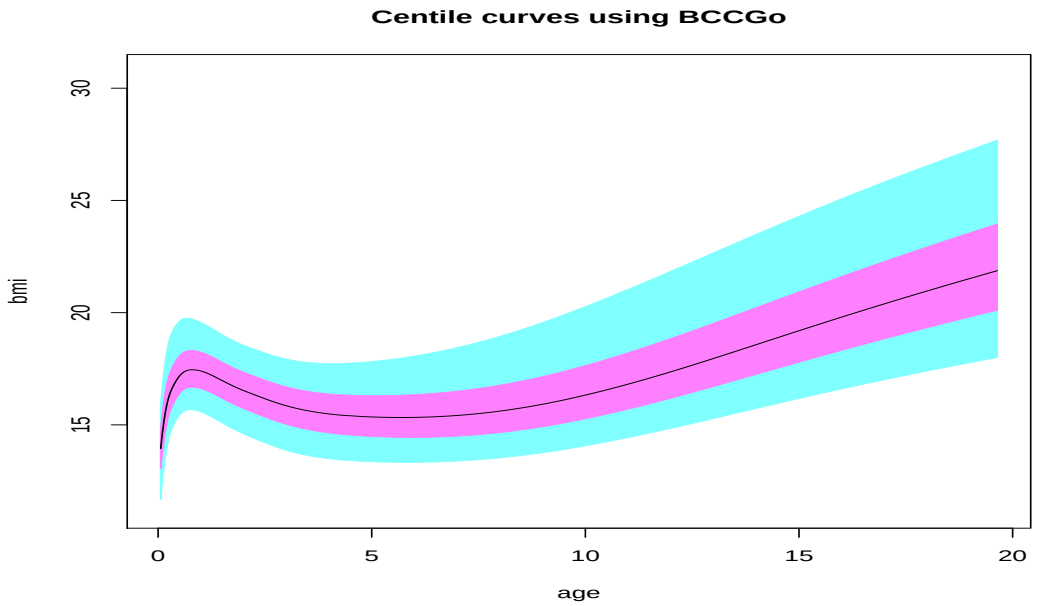


Figure 13.10: A fan-chart (centile) curves using Box-Cox Cole and Green distribution for the sampled 1000 observation from the `dbbmi` data

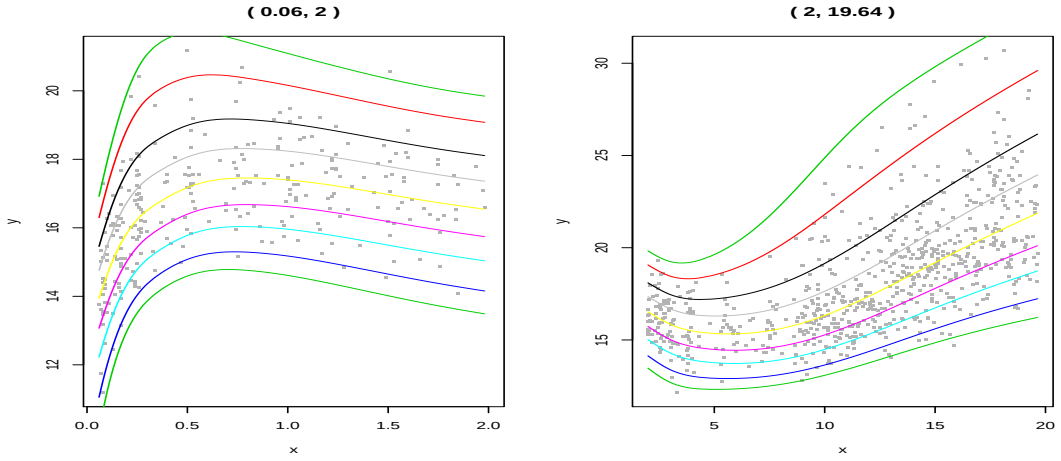
```
## 75      74.254    75.0000
## 90      90.299    89.6175
## 98      97.761    97.8142
## 99.6    99.627   100.0000
```

see Figure 13.11 for the plot.

The Table above gives the sample % of cases below the 0.4,2,10,...,99.6 centile curves for each of the two age ranges in the split, i.e. age range (0.03 to 2) and age range (2 to 21.7), [where 0.03 and 21.7 are the minimum and maximum ages in the data set].

The arguments for the function `centiles.split` are

`obj`	a fitted `gamlss` object
`xvar`	the unique explanatory variable
`xcut.points`	the x-axis cut off point(s) e.g. `c(20,30)`. If `xcut.points=NULL` then the `n.inter` argument is activated
`n.inter`	if `xcut.points=NULL` this argument gives the number of intervals in which the x-variable will be split, with default value 4
`cent`	a vector with elements the % centile values for which the centile curves have to be evaluated



R code on page 327

Figure 13.11: Two centiles curves using Box-Cox Cole and Green distribution to the sample of 1000 observations from the BMI data

`legend`	whether a legend is required in the plots or not, the default is `legent=FALSE`
`ylab`	the y-variable label
`xlab`	the x-variable label
`overlap`	how much overlapping in the `xvar` intervals. Default value is 'overlap=0' for non overlapping intervals
`save`	whether to save the sample percentages or not, with default equal to 'TRUE'. In this case the function produces a matrix giving the sample percentages for each interval
`plot`	whether to plot the centiles. This option is useful if the sample statistics only are to be used
`...`	for extra arguments in the `par()` plotting function

For example a four equal number of observation split of age x can be achieved using:

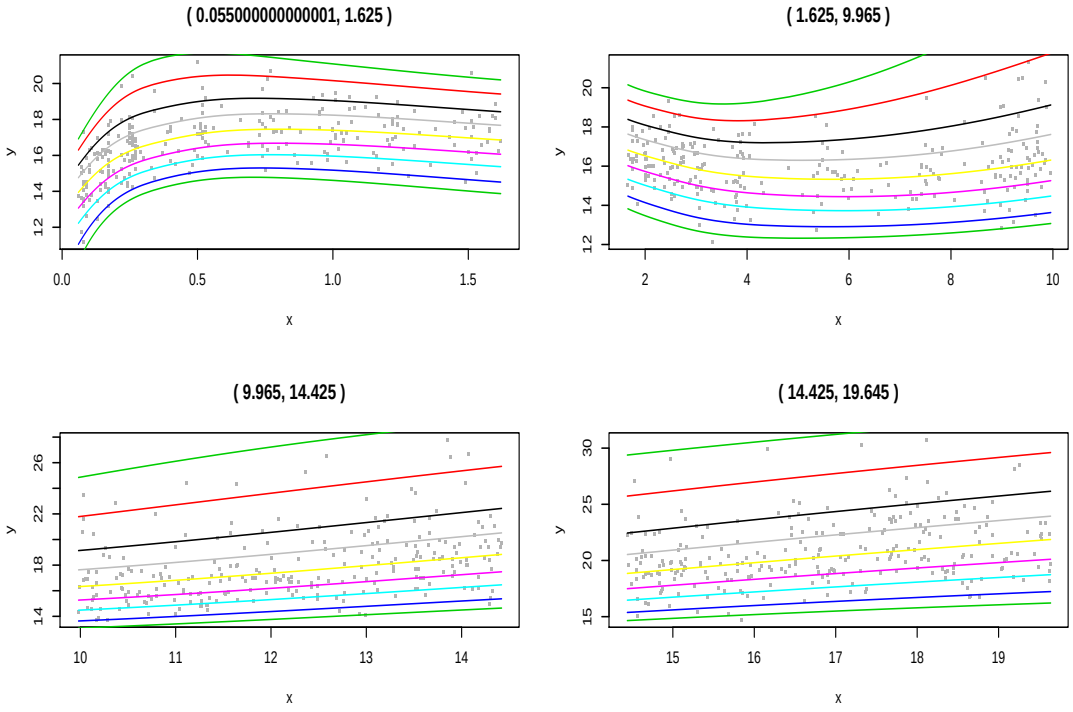
Figure 13.12

```
centiles.split(m0,dbbmi1$age)
```

```
##      0.05500 to 1.625 1.625 to 9.965 9.965 to 14.425 14.425 to 19.645
## 0.4               0.0            0.4             0.4              0.4
## 2                 2.8            2.8             1.6              2.8
## 10               10.8            9.6            11.6             10.4
## 25               23.6           24.0            28.0             22.4
```

```
## 50                48.0          50.0          52.4          49.2
## 75                74.0          74.8          77.6          72.8
## 90                89.6          90.0          89.2          90.4
## 98                97.6          99.2          96.4          98.0
## 99.6              99.6         100.0         100.0         100.0
```

see Figure 13.12 for the plot.



R code on page 329

Figure 13.12: Centiles curves for four age ranges using Box-Cox Cole and Green distribution for the BMI data

Sample centile statistics for different values of the x-variable can be obtained by suppressing the plot using the argument `plot=FALSE`. For example in order to get sample statistics in 6 age ranges with equal numbers of observations use

```
centiles.split(m0, xvar=dbbmi1$age, n.inter=6, plot=FALSE)
```

```
##      0.05500 to 0.87499 0.87500 to 3.035 3.035 to 9.965 9.965 to 13.095
## 0.4            0.000000         0.000000      0.5988024       0.5988024
## 2              2.994012         1.807229      3.5928144       1.7964072
## 10            10.778443         9.036145     10.7784431      12.5748503
## 25            25.149701        22.289157     23.9520958      32.3353293
## 50            49.101796        48.795181     49.1017964      58.0838323
```

```
## 75            75.449102        72.289157     75.4491018      79.6407186
## 90            89.820359        90.963855     88.6227545      89.2215569
## 98            97.005988        98.192771    100.0000000      96.4071856
## 99.6         100.000000        99.397590    100.0000000     100.0000000
##      13.095 to 15.865 15.865 to 19.645
## 0.4          0.6024096         0.000000
## 2            3.0120482         1.796407
## 10          10.2409639        10.179641
## 25          20.4819277        22.754491
## 50          44.5783133        49.700599
## 75          75.9036145        70.059880
## 90          90.3614458        89.820359
## 98          96.9879518        98.203593
## 99.6       100.0000000       100.000000
```

13.9 The function `centiles.com()`

This function is useful comparing centile curves produced by different fitted models. Here we fit a new `lms` object using the SHASH distribution and compare the result with the original `lms` model `m0` fitted in Setion 13.5).

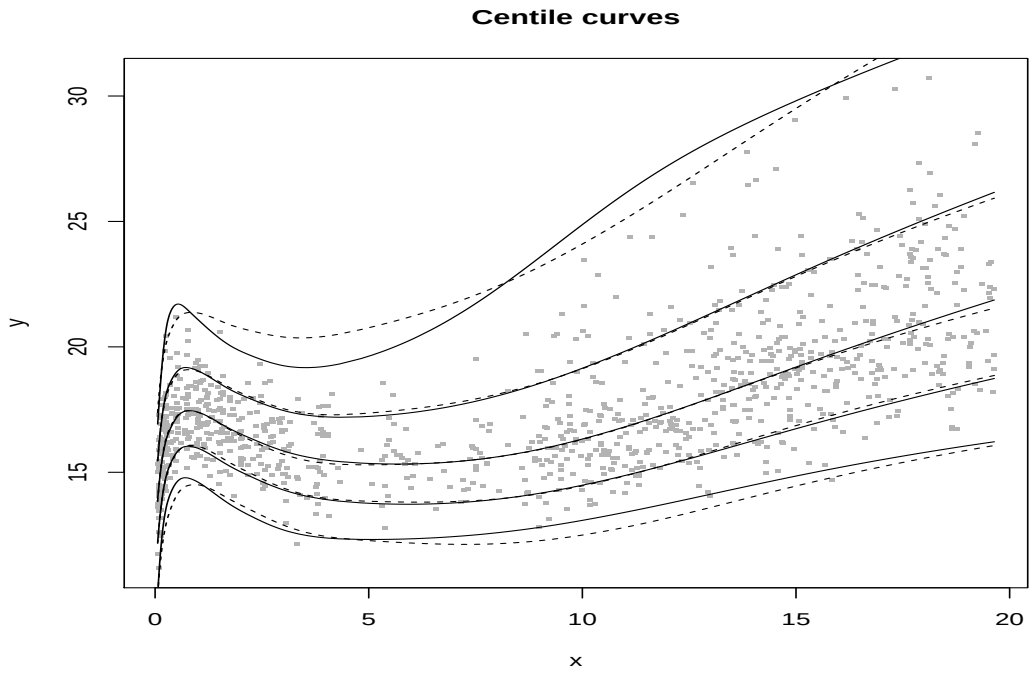
```
m2 <-  lms(bmi, age, data=dbbmi1, trans.x=TRUE , families=c("SHASH"))
centiles.com(m0, m2, xvar=dbbmi1$age, legend=FALSE, color=FALSE)
```

```
## *** Checking for transformation for x ***
## *** power parameters  0.02764176 ***
## *** Initial  fit***
## GAMLSS-RS iteration 1: Global Deviance = 4261.962
## . . .
## % of cases below  90.97766 centile is  90.8
## % of cases below  97.88226 centile is  97.7
## % of cases below  99.50523 centile is  99.6
```

```
centiles.com(m0, m2, xvar=dbbmi1$age, legend=FALSE, color=FALSE)
```

Figure 13.13

```
## ********  Model 1 ********
## % of cases below  0.4 centile is  0.3
## % of cases below  10 centile is  10.6
## % of cases below  50 centile is  49.9
## % of cases below  90 centile is  89.8
## % of cases below  99.6 centile is  99.9
## ********  Model 2 ********
## % of cases below  0.4 centile is  0.2
## % of cases below  10 centile is  11
## % of cases below  50 centile is  49.6
## % of cases below  90 centile is  89.4
## % of cases below  99.6 centile is  99.7
```



R code on page 331

Figure 13.13: Comparison of centiles curves using the BCCGo (Box-Cox Cole and Green) and SHASH (Sinh-Arcsinh) distributions

Most of the arguments of the function are similar to the ones in `centiles()`. Here we highlight the argument `no.data` useful for excluding data points from the plot.

13.10 The functions `centiles.pred()` and `z.scores()`

The `centiles.pred()` is designed to create predictive centiles curves for new x-values, given a `gamlss` fitted model. The function has three different functionalities which are decribed below:

case 1: For given new x-values and given percentage centiles, calculates a matrix containing the centile values for y.

case 2: For given new x-values and standard normalized centile values, calculates a matrix containing the centile values for y.

case 3: For given new x-values and new y-values calculates the z-scores [one Z-score for each (x, y) pairs].

The first two options are useful for creating growth curve tables and plots useful for publication purposes. The third option is useful for checking where new observations are lying within the standard growth charts. Because of the importance of this latest task the function `z.scores()` is created to provide the same functionality. As with all the rest of the functions in this chapter, the functions `centiles.pred()` and `z.scores()` apply to models with only one explanatory variable.

case 1

To demonstrate the first case above we start by creating new values for `age`, `newx<-seq(0,2,.05)` and use them to find the corresponding centiles which are stored in a matrix `mat`. The centiles are created at the default centile % values of `c(0.4, 2, 10, 25, 50, 75, 90, 98, 99.6)`. These centiles then can be plotted using the `centiles.pred` argument `plot=TRUE`. Note that we can use the model fitted by `lms()`.

```
newage<-seq(0,2,.05)
mat <- centiles.pred(m0, xname="age", xvalues=newage)
head(mat)
```

```
##    age   C0.4     C2   C10   C25   C50   C75   C90   C98 C99.6
## 1 0.00  7.467  9.329 10.99 11.97 12.87 13.64 14.24 14.93 15.41
## 2 0.05  9.690 10.789 12.02 12.88 13.76 14.57 15.25 16.07 16.66
## 3 0.10 11.211 11.993 12.99 13.77 14.62 15.47 16.23 17.19 17.93
## 4 0.15 12.263 12.903 13.77 14.49 15.33 16.21 17.03 18.14 19.04
## 5 0.20 13.002 13.566 14.36 15.04 15.86 16.75 17.64 18.88 19.95
## 6 0.25 13.514 14.036 14.78 15.43 16.24 17.15 18.07 19.41 20.62
```

```
mat <- centiles.pred(m0, xname="age", xvalues=newage, plot=TRUE,
                     legend=FALSE, ylab="BMI", xlab="age")
```

Figure 13.14

R code on page 333

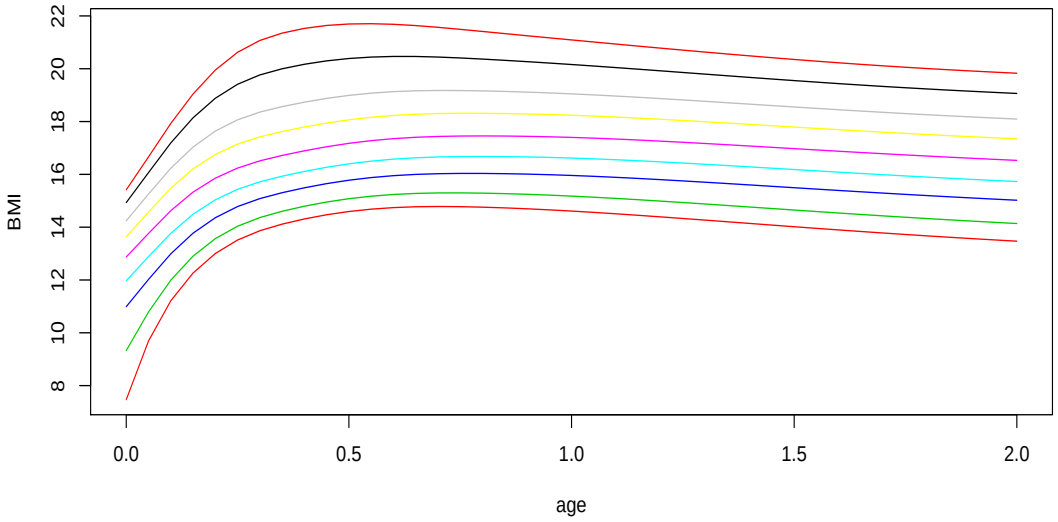


Figure 13.14: A plot of centiles curves in the age range 0 to 2 using selected % centiles

case 2

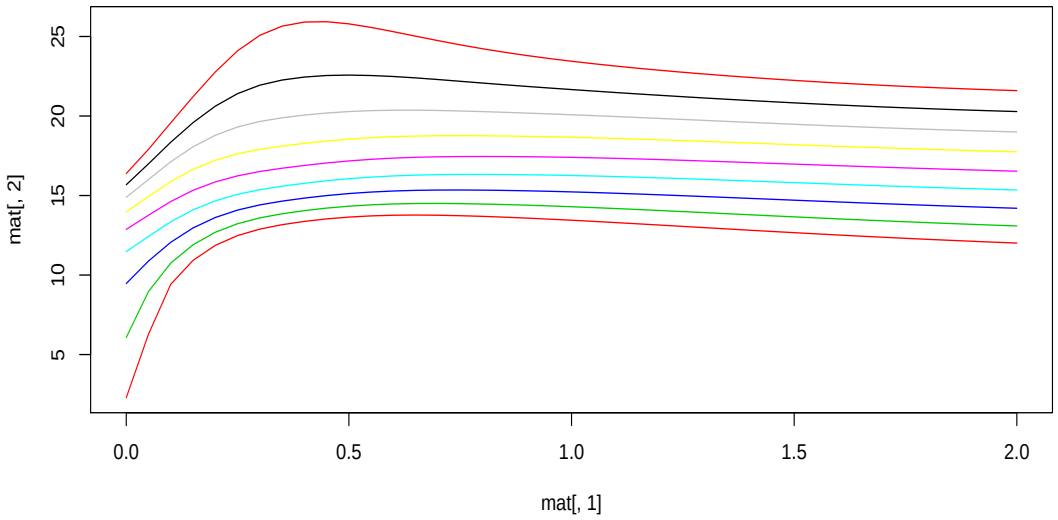
In the second case the objective is to create centiles based not on percentages but on standard normalized values or Z value. These are using the `centiles.pred` argument `dev` with default Z values `dev=c(-4, -3, -2, -1, 0, 1, 2, 3, 4)`. [Note that the corresponding centile percentages for the standard normalized values can be obtained by applying $\Phi^{-1}() = qNO()$, the inverse cumulative distribution function of a standard normal distribution, i.e $\% = \Phi^{-1}(z)$]. [The resulting %'s are $(0.003, 0.135, 2.275, 15.866, 50, 84.134, , 97.725, 99.865, 99.997)$.] We use the same new `age` values as above but this time we use the argument `type="standard-centiles"`.

Figure 13.15

```
mat  <- centiles.pred(m0, xname="age",xvalues=newage,
                      type="standard-centiles")
head(mat)
```

```
##    age     -4     -3     -2    -1     0     1     2     3     4
## 1 0.00  2.288  6.075  9.466 11.47 12.87 13.97 14.89 15.68 16.38
## 2 0.05  6.286  8.967 10.881 12.43 13.76 14.94 16.01 17.00 17.91
## 3 0.10  9.421 10.753 12.062 13.35 14.62 15.88 17.12 18.35 19.57
## 4 0.15 10.922 11.904 12.962 14.10 15.33 16.65 18.06 19.58 21.22
## 5 0.20 11.867 12.692 13.619 14.67 15.86 17.22 18.79 20.62 22.77
## 6 0.25 12.483 13.230 14.085 15.08 16.24 17.63 19.31 21.41 24.10
```

```
mat <- centiles.pred(m0, xname="age", xvalues=newage, type="s",
                     dev = c(-4, -3, -2, -1, 0, 1, 2, 3, 4),
                     plot = TRUE, legend=FALSE )
```



R code on page 334

Figure 13.15: A plot of prediction centiles curves using selected standard normalized deviates (i.e. Z values)

case 3

Case 3 is when we are in a situation in which a new individual is available for whom we know the value of the y-variable (say BMI), and his/her age and we want to classify whether they are on risk. This is done by obtaining the z-score of the indivisual. Z-scores below -2 and above 2 are usually of concern. Here we show how to obtain the z-scores using the `centiles.pred()` and the `z.scores()` respectively.

```
centiles.pred(m0, xname="age", xvalues=c(2,5,10,15), yval=c(20,18,25,14),
              type="z-scores")
```

```
## [1]  2.784040  1.752878  2.672242 -3.428404
```

```
z.scores(m0, x=c(2,5,10,15), y=c(20,18,25,14))
```

```
## [1]  2.784040  1.752878  2.672242 -3.428404
```

The `z.scores()` function is simpler to use in this case.

The arguments for the function `centiles.pred` are

obj — a fitted `gamlss` object

type — the default, `centiles`, gets the centiles values given in the option `cent`. `type="standard-centiles"` gets the standard centiles given in the `dev`.

`type="z-scores"` gets the z-scores for given y and x new values

xname	the name of the unique explanatory variable (it has to be the same as in the original fitted model)
xvalues	the new values for the explanatory variable where the prediction will take place
power	if power transformation is needed
yval	the response values for a given x values required for the calculation of `z-scores`
cent	a vector with elements the % centile values for which the centile curves have to be evaluated
dev	a vector with elements the standard normalized deviate values (or Z values) for which the centile curves have to be evaluated in the option `type="standard-centiles"`
plot	whether to plot the "centiles" or the "standard-centiles", the default is `plot=FALSE`
legend	whether a legend is required in the plot or not, the default is `legend=TRUE`
...	for extra arguments

The `z.scores()` function has only three arguments, i) `object` for fitted `lms` model ii) `y` for new y values and iii) `x` for new x values.

13.11 Quantile Sheets using the function `quantSheet()`

In this Section we are describe the use of quantile sheets regression for constructing growth curves. Quantile sheets were developed by Schnabel and Eilers [2013a,b], in order to overcame some of the problems associated with quantile regression. In particular the main advance of the quantile sheets is the simultaneous estimation of the quantiles, and the introduction of a smoothing parameter in the the response variable direction. This reduce the wide variability of the quantile curves and make them look more realistic. It also avoids (but do not eliminate completely) the problem of crossing quantiles.

Quantile sheets can be fitted within the **gamlss** packages using the function `quantSheets()`. The function is a modified version of an earlier **R** function given to the authors by Paul Eilers. In its current form it can take only one explanatory variable. The function is fast compared to `lms)` even for large data sets.

Smoothing parameters

As was mention earlier the fit of quantiles/centiles depends on two smoothing parameters:

1. the `x.lambda`, smoothing parameters in the direction of the x-variable and
2. the `p.lambda`, smoothing parameter in the direction of the response variable (or more precise its probability).

The smoothing parameters in `quantSheets()` are **not** estimated automatically and they should be chosen by the user either by inspection or by some other criteria. Unfortunately since quantile sheets estimation do not easily provide an overall measurement of fit, e.g. GAIC, the selection of the two smoothing parameters has to be do by other means. Here we use propose a heuristic

methods of choosing the smoothing parameters based on residuals, where the adequacy of the model is checked throughout residual diagnostics.

Residuals

The calculation of the normalised quantile residuals or z-scores within a quantile regression is not straightforward, as when a parametric distribution is assumed. The following approach is used here to define the z-scores.

The fitted `quantSheet` model provides for each district explanatory x-variable value fitted quantiles. Given the fitted quantile values, a non-parametric distribution can be constructed for each district point, using the **gamlss** function `FlexDist()`. The function `FlexDist()` constructs a (non-parametric) distribution using known quantile or expectile values of the distribution, and provides numerically calculated probability density functions (pdf) and commutative density functions (cdf) functions for the distribution. Given now the estimated cdf at value x, the probability integral transform (PIT) residuals can be found, `pit=cdf(y)`, and therefore the normalised quantile residuals `qnorm(pit)`. The function `z.scoresQS()` performs those steps. Note that because the cdf function is different for each distinct values of the explanatory variable x for large data sets the calculation of the quantile residuals can take several minutes. To avoid this problem the function `residuals.quantSheets()` provides, as a default, a quicker way of calculating the residuals. It starts by binning the observations in the x-direction, to say 100 intervals (option `inter=100`), with equal number of observations at each bin. It then calculates (using `FlexDist()`) the cdf at the middle points of the intervals and evaluates the PIT's and quantile residuals of all the observations that fall in the bin. This reduces the time for calculating the quantile intervals considerably. The full quantile residuals can be obtained using the option `all=TRUE`.

While the approach described above seems to work well the user should be aware that the the function `flexDist()` used to construct the pdf and cdf functions is using penalties which themselves depend on smoothing parameters. While the defaults smoothing parameters seem to work well a visual inspection at least at some district values of the x is recommended.

fitting the model

Here we fit a quantile sheets model on the 1000 sampled observations from the `dbbmi` data. We first use the function `findPower()` to find a suitable power transformation for x and use it in the option `power` in the quantile sheets fitting. The smoothing parameters values `x.lambda = 1` and `p.lambda = 10` are choose arbitrary.

Figure 13.16

```
ppp<-findPower(dbbmi1$bmi,dbbmi1$age)
```

```
## *** Checking for transformation for x ***
## *** power parameters  0.3295 ***
```

```
qs1<-quantSheets(bmi, age, data = dbbmi1,
                 cent = c(0.4, 2, 10, 25, 50, 75, 90, 98, 99.6),
                 x.lambda = 1, p.lambda = 10, logit = TRUE, power = ppp)
```

```
## % of cases below  0.4 centile is  0
## % of cases below  2 centile is  0.4
```

```
## % of cases below  10 centile is  8.4
## % of cases below  25 centile is  27.4
## % of cases below  50 centile is  53.8
## % of cases below  75 centile is  75.9
## % of cases below  90 centile is  88.7
## % of cases below  98 centile is  96.8
## % of cases below  99.6 centile is  98.6
```

The fitted centiles are shown in figure 13.16.

R code on page 337

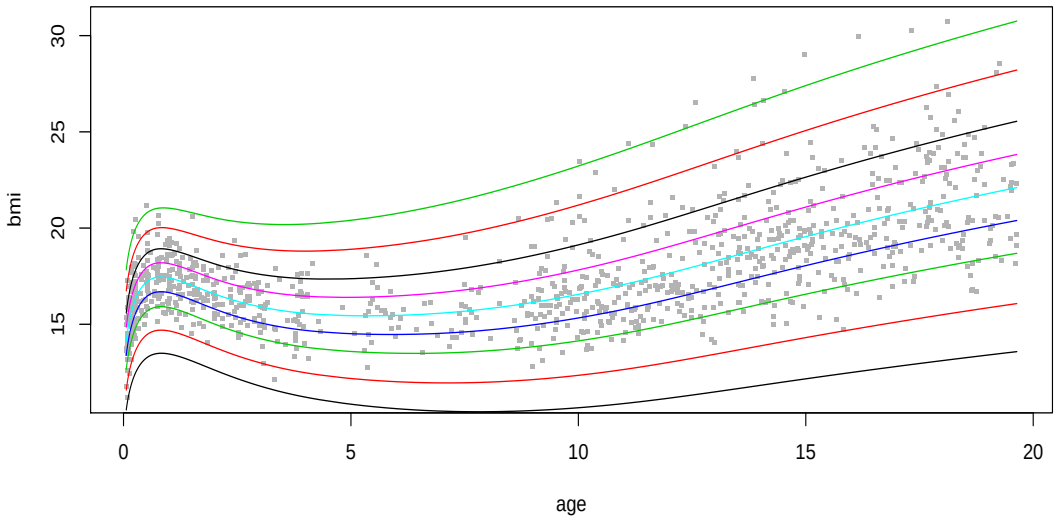


Figure 13.16: Quantile sheet curves fitted to the the sample of the `dbmbi` data using smoothing parameters `x.lambda = 1` and `p.lambda = 10`

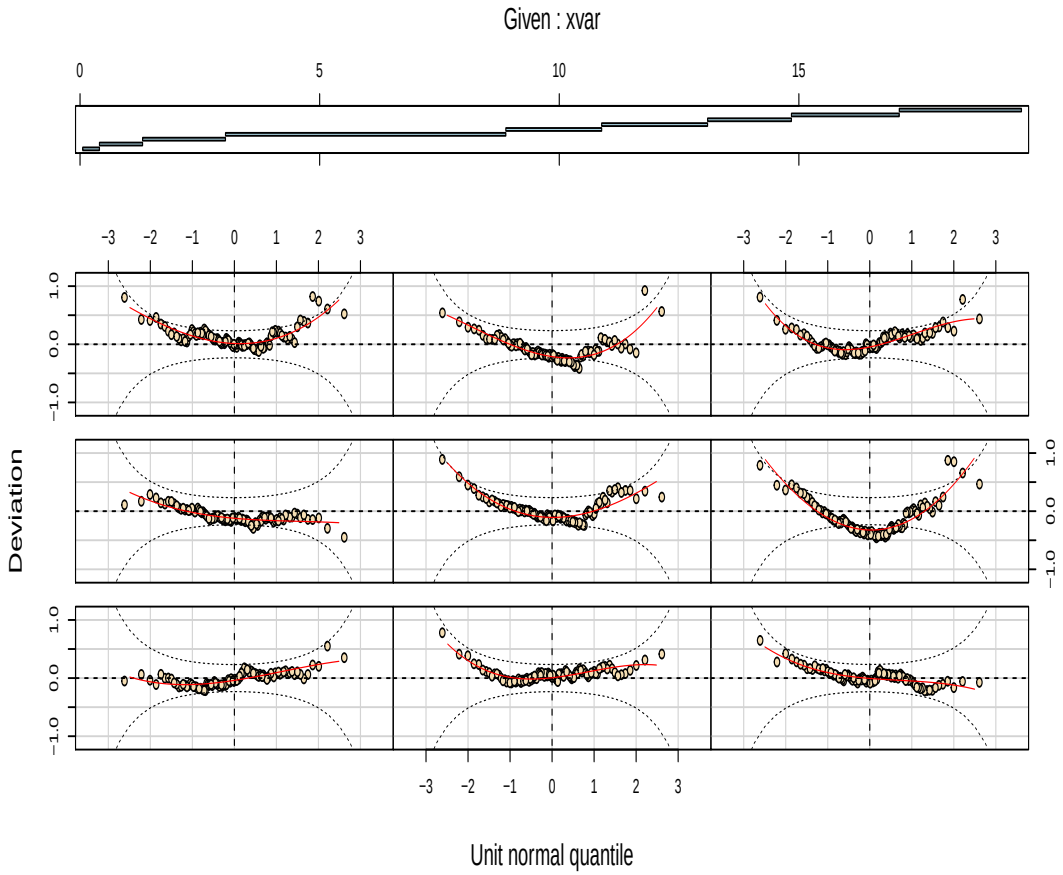
Figure 13.17

To check the model we first calculate the residuals and then we use worm plot and Q-statistics as diagnostics.

```
res1 <- resid(qs1)
wp(resid=res1, xvar=dbbmi1$age, n.inter = 9 )
```

```
## number of missing points from plot= 0  out of  112
## number of missing points from plot= 0  out of  111
## number of missing points from plot= 0  out of  110
## number of missing points from plot= 0  out of  112
## number of missing points from plot= 0  out of  111
## number of missing points from plot= 0  out of  111
## number of missing points from plot= 0  out of  112
## number of missing points from plot= 0  out of  110
```

```
## number of missing points from plot= 0  out of  111
#round(Q.stats(resid=res1, xvar=dbbmi1£age), 3)
```



R code on page 338

Figure 13.17: Worm plots from the Quantile sheet curves fitted to the sample of `dbmbi` using smoothing parameters `x.lambda = 1` and `p.lambda = 10`

```
round(Q.stats(resid=res1, xvar=dbbmi1$age), 3)
```

```
##                       Z1     Z2    Z3     Z4
##  0.055 to  0.285 -0.840 -0.500 2.630  1.139
##  0.285 to  1.155 -0.281 -0.062 1.676 -1.066
##  1.155 to  2.525 -0.337 -0.968 2.680  0.314
##  2.515 to  5.915  0.267 -0.631 4.092  0.873
##  5.975 to  9.965 -0.403 -0.888 4.776  1.512
##  9.975 to 11.705  0.447  0.924 5.456  1.941
## 11.715 to 13.605 -0.048  0.555 4.897  1.800
## 13.615 to 15.135  1.149  0.354 3.342  0.894
```

```
## 15.135 to 17.385  0.351  0.264 2.659  0.101
## 17.405 to 19.645  0.892  0.509 1.291 -1.279
```

Both diagnostic plots, Figure 13.17 for the worm plots and the left plot of Figure 13.20 for the Q-statistics, show evidences that the skewness of the distribution of response is not modelled properly. For example, the worm plots show quadratic shapes while the skewness column of the Q-statistics, `Z3`, has values larger than 2. From the two smoothing parameters, the `p.lambda` is the one that it is most likely to effect the shape of the distribution of the response variable so the next step is to decrees the value `p.lambda` while simultaneously checking the `Z3` column of the Q-statistics. The following combination of smoothing parameters seems that it is working well.

Figure 13.18

```
qs2<-quantSheets(bmi, age, data = dbbmi1,
                 cent = c(0.4, 2, 10, 25, 50, 75, 90, 98, 99.6),
                 x.lambda = 1, p.lambda = .05, logit = TRUE, power = ppp)
```

```
## % of cases below  0.4 centile is  0.4
## % of cases below  2 centile is  2
## % of cases below  10 centile is  10.7
## % of cases below  25 centile is  25.7
## % of cases below  50 centile is  50
## % of cases below  75 centile is  74.4
## % of cases below  90 centile is  89.7
## % of cases below  98 centile is  97.9
## % of cases below  99.6 centile is  99.6
```

It seems that decreasing `p.lambda` to 0.05 provides a model with good residual diagnostics as shown in Figure 13.19 and the right plot of Figure 13.20.

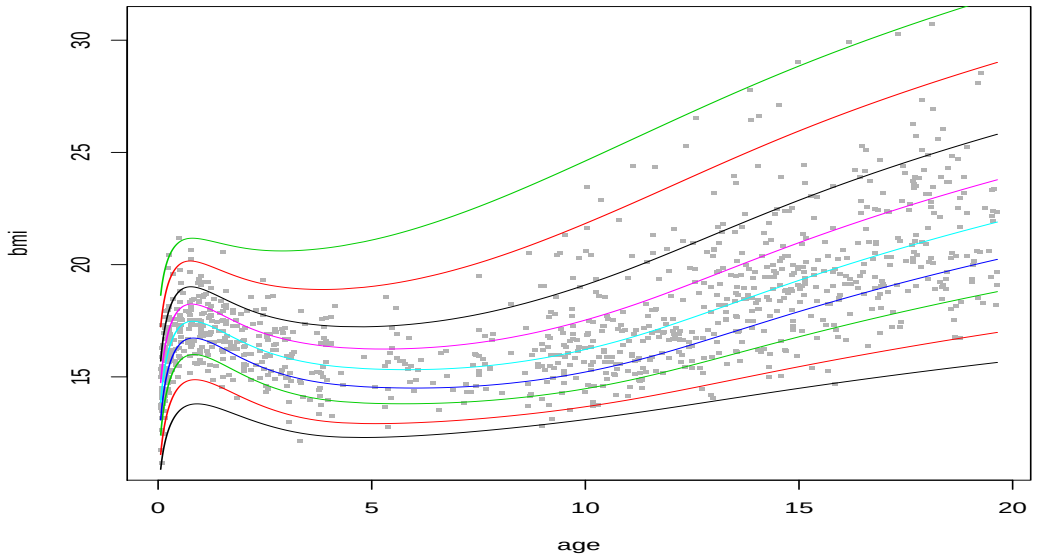
Figure 13.19

```
res2 <- resid(qs2)
wp(resid=res2, xvar=dbbmi1$age, n.inter = 9 )
```

```
## number of missing points from plot= 0  out of  112
## number of missing points from plot= 0  out of  111
## number of missing points from plot= 0  out of  110
## number of missing points from plot= 0  out of  112
## number of missing points from plot= 0  out of  111
## number of missing points from plot= 0  out of  111
## number of missing points from plot= 0  out of  112
## number of missing points from plot= 0  out of  110
## number of missing points from plot= 0  out of  111
```

```
round(Q.stats(resid=res2, xvar=dbbmi1$age),3)
```

```
##                      Z1     Z2     Z3     Z4
##  0.055 to  0.285 -0.872 -0.360  0.279  0.201
##  0.285 to  1.155 -0.292  0.513 -0.814 -1.788
##  1.155 to  2.525 -0.129 -0.402 -0.766 -0.497
##  2.515 to  5.915  0.630 -0.455  0.562 -0.255
##  5.975 to  9.965 -0.074 -0.514  0.616  0.393
##  9.975 to 11.705  0.608  0.483  2.169  0.444
```



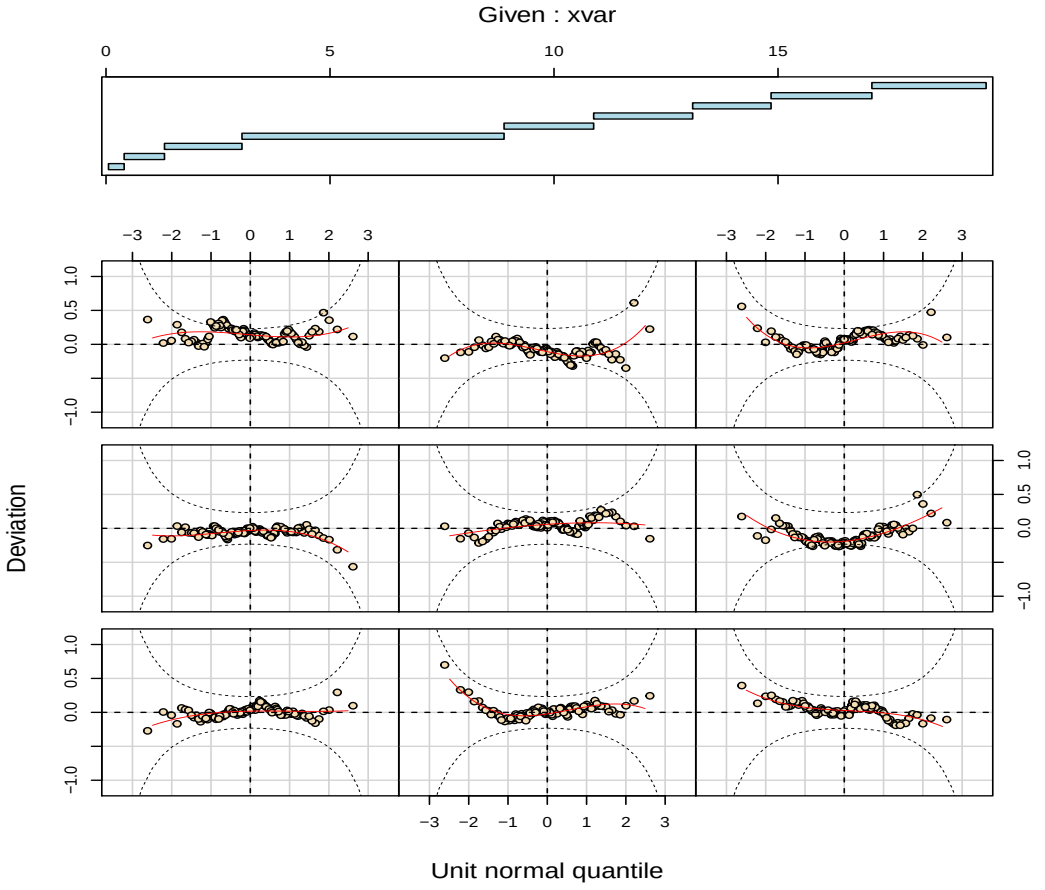
R code on page 340

Figure 13.18: Quantile sheet curves fitted to the sample of `dbmbi` data using smoothing parameters `x.lambda = 1` and `p.lambda = .05`

```
## 11.715 to 13.605  0.136  0.235  1.725  0.423
## 13.615 to 15.135  1.274  0.019  0.496  0.103
## 15.135 to 17.385  0.398  0.047  0.609 -0.552
## 17.405 to 19.645  0.908  0.257  0.090 -1.714
```

```
op<- par(mfrow=c(1,2))
Q.stats(resid=res1, xvar=dbbmi1$age )

##                         Z1       Z2    Z3      Z4
##  0.055 to  0.285 -0.84038 -0.50042 2.630  1.1390
##  0.285 to  1.155 -0.28143 -0.06233 1.676 -1.0663
##  1.155 to  2.525 -0.33731 -0.96758 2.680  0.3138
##  2.515 to  5.915  0.26732 -0.63061 4.092  0.8728
##  5.975 to  9.965 -0.40328 -0.88758 4.776  1.5123
##  9.975 to 11.705  0.44664  0.92449 5.456  1.9412
## 11.715 to 13.605 -0.04814  0.55475 4.897  1.7997
## 13.615 to 15.135  1.14900  0.35356 3.342  0.8941
## 15.135 to 17.385  0.35094  0.26414 2.659  0.1015
## 17.405 to 19.645  0.89210  0.50939 1.291 -1.2789

Q.stats(resid=res2, xvar=dbbmi1$age )

##                         Z1       Z2       Z3      Z4
##  0.055 to  0.285 -0.87208 -0.35978  0.27862  0.2011
```

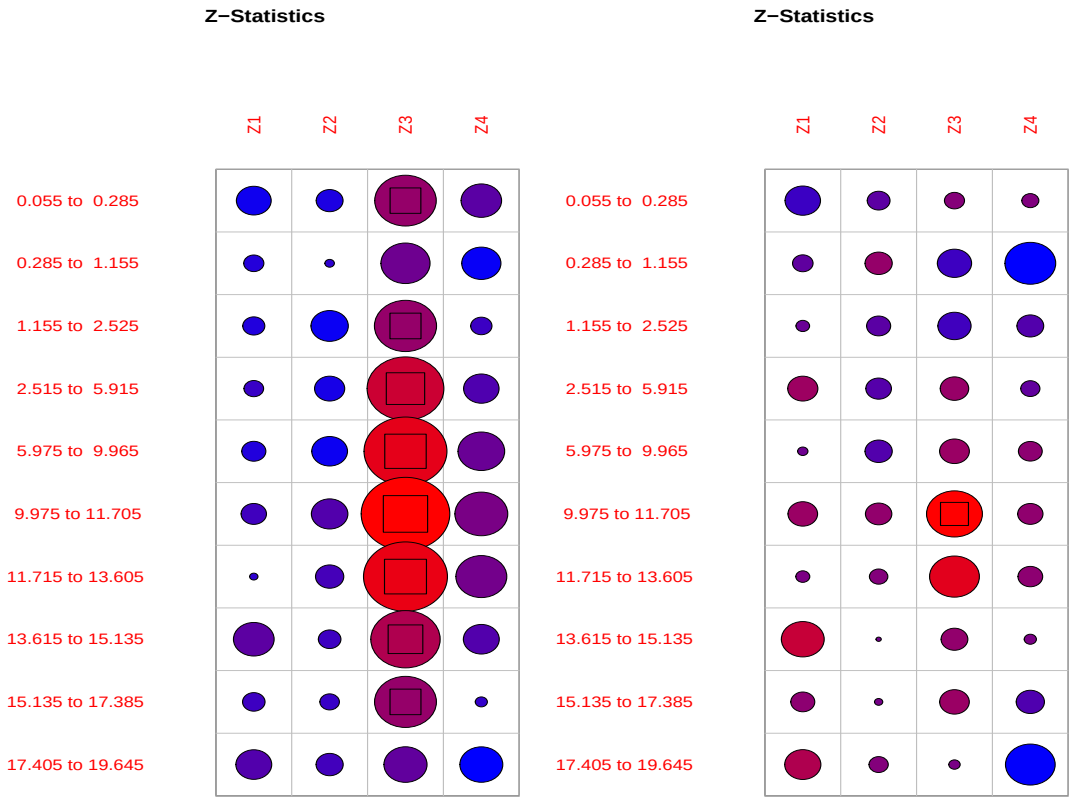
Figure 13.20



R code on page 340

Figure 13.19: Worm plots from the fitted quantile sheet to the sample of the `dbmbi` data using smoothing parameters `x.lambda = 1` and `p.lambda = 0.05`

```
##  0.285 to  1.155 -0.29166  0.51284 -0.81378 -1.7884
##  1.155 to  2.525 -0.12853 -0.40202 -0.76565 -0.4971
##  2.515 to  5.915  0.63039 -0.45546  0.56220 -0.2548
##  5.975 to  9.965 -0.07361 -0.51423  0.61587  0.3927
##  9.975 to 11.705  0.60808  0.48312  2.16861  0.4435
## 11.715 to 13.605  0.13556  0.23523  1.72463  0.4233
## 13.615 to 15.135  1.27433  0.01890  0.49641  0.1027
## 15.135 to 17.385  0.39846  0.04702  0.60892 -0.5521
## 17.405 to 19.645  0.90795  0.25705  0.08956 -1.7138

par(op)
```



R code on page 341

Figure 13.20: Q-statistics plots from the two fitted quantile sheets models to the sample of the `dbmbi` data using smoothing parameters: i) `x.lambda = 1` and `p.lambda = 10` left plot and ii) `x.lambda = 1` and `p.lambda = 0.05` respectively

Chapter 14

Further Applications

This chapter provides further applications of GAMLSS modelling. In particular:

- the species data as an example fitting different discrete count distributions to data
- the hospital stay data as an example of fitting binomial type distribution.
- the film data: as an example of fitting smoothing two dimensional surfaces on a continuous response variable.

14.1 Count data: the fish species data

Data summary: the fish species data
R data file: `species` in package **gamlss.dist** of dimensions 70×2
variables
`fish` : the number of different species in 70 lakes in the world
`lake` : the lake area
purpose: to demonstrate the fitting of count data distributions

The number of different fish species (`fish`) was recorded for 70 lakes of the world together with explanatory variable `x=log(lake)`, i.e. `x` = log lake area. The data are plotted in Figure 14.1.

```
library(gamlss)
data(species)
# creating the log(lake)
species$x <- log(species$lake)
plot(fish~x,data=species, col="blue")
```

Figure 14.1

The data are given and analysed by Stein and Juritz (1988) using a Poisson inverse Gaussian, $PIG(\mu, \sigma)$ distribution for `fish` with a linear model in `log(lake)` for $\log \mu$ parameter and a constant for $\log \sigma$.

Rigby *et al.* (2008), when analysing this data set, identified the following questions that need to

R code on page 345

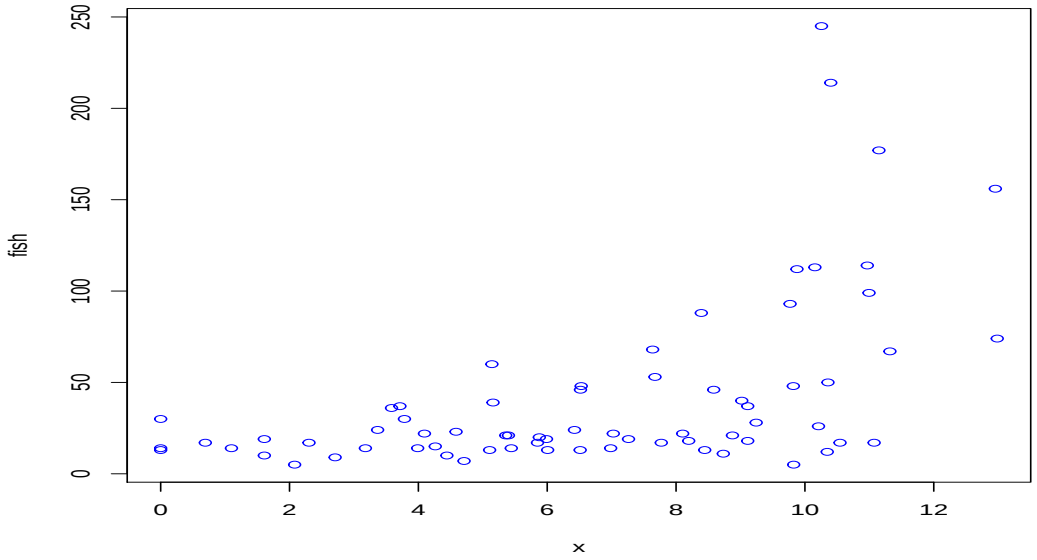


Figure 14.1: The fish species data

be answered. Note that the same questions could apply to any regression type situation where the response variable is counts and where `x` represents a set explanatory variables.

- How does the mean of the response variable depend on `x`?
- Is the response variable overdispersed Poisson?
- How does the variance of the response variable depend on its mean?
- What is the distribution of the response variable given `x`?
- Do the scale and shape parameters of the response variable distribution depend on `x`?

Here we will model the data using different discrete distributions and consider flexible models for the distributional parameters, where any or all of them can possibly depend on the explanatory variable `log(lake)`.

We start by fitting seven different count distributions to the data [Poisson (PO), double Poisson (DPO), negative binomial type I and II (NBI, NBII), Poisson inverse Gaussian (PIG), Delaporte (DEL) and Sichel (SICHEL)] using first a linear and then a quadratic polynomial in `x=log(lake)`. The AIC of each model is then printed for comparison. Note that the `GAIC(...,k=2)` corresponds to the AIC and since is the default value can be omitted.

```
# the count distributions
fam<-c("PO","DPO", "NBI", "NBII", "PIG", "DEL", "SICHEL")
#creating lists to keep the results
m.l<-m.q<-list()
```

```
# fitting the linear in x models
for (i in 1:7) {
m.l[[fam[i]]]<-GAIC(gamlss(fish~x,data=species, family=fam[i], n.cyc=60,
                     trace=FALSE),k=2)}
# fitting the quadratic in x models
for (i in 1:7) {
m.q[[fam[i]]]<-GAIC(gamlss(fish~poly(x,2),data=species, family=fam[i],
                           n.cyc=60, trace=FALSE), k=2)}
# print the AIC's
unlist(m.l)
```

```
##        PO       DPO       NBI      NBII       PIG       DEL    SICHEL
## 1900.1562  654.1616  625.8443  647.5359  623.4632  626.2330  625.3923
```

```
unlist(m.q)
```

```
##        PO       DPO       NBI      NBII       PIG       DEL    SICHEL
## 1855.2965  655.2520  622.3173  645.0129  621.3459  623.5816  623.0995
```

The Poisson model has a very large AIC compared to the rest of the distributions so we can conclude that the data are overdispersed. The quadratic polynomial in `x` seems to fit better than the linear term across the different count distributions (except for DPO). The best model at this stage is the Poisson inverse Gaussian (PIG) model with a quadratic polynomial in `x`. We now compare the AIC of a PIG model with a cubic smoothing spline in `x` instead of a quadratic polynomial in `x`. The total "effective" degrees of freedom for `x` in the default cubic spline model (including the constant and linear term) is 5 compared to 3 in the quadratic model.

```
GAIC(gamlss(fish~cs(x), data=species, family=PIG, trace=FALSE))
```

```
## [1] 623.9328
```

The cubic smoothing spline does not seem to improve the model, so we keep the quadratic polynomial in `x`. We shall now try to model $\log(\sigma)$ as a linear function of `x` in the six remaining count distributions.

```
# redefine the list of distributions
fam<-c("DPO","NBI", "NBII", "PIG", "DEL", "SICHEL")
m.ql<-list()
for (i in 1:6) {
m.ql[[fam[i]]]<-GAIC(gamlss(fish~poly(x,2),data=species,
                    sigma.fo=~x, family=fam[i], n.cyc=60, trace=FALSE))}
unlist(m.ql)
```

```
##      DPO      NBI     NBII      PIG      DEL   SICHEL
## 626.4056 614.9565 615.1250 612.3667 614.6059 613.7327
```

Modelling $\log(\sigma)$ as a linear function of `x` improves the AIC for all models. The PIG model is still the "best". Since the Sichel and the Delaporte distributions have three parameters we will try to model the third parameter ν as a linear function of `x`. The Sichel uses the `identity` as the default link for ν while the Delaporte uses the `logit`.

```
fam<-c("DEL", "SICHEL")
m.qll<-list()
```

```
for (i in 1:2) {
m.qll[[fam[i]]]<-GAIC(gamlss(fish~poly(x,2),data=species,
                    sigma.fo=~x, nu.fo=~x, family=fam[i], n.cyc=60,
                    trace=FALSE))}
unlist(m.qll)
```

```
##      DEL   SICHEL
## 614.7376 611.6346
```

Modelling the ν as a linear function of `x` improves the Sichel model (which now has lower AIC than the PIG model) but not the Delaporte model. A further simplification of the Sichel model can be achieved by dropping the linear terms in `x` for the $\log(\sigma)$ model which given the linear model in `x` for ν does not seem to contribute anything to the fit (a least according to the AIC):

```
GAIC(gamlss(fish~poly(x,2),data=species, sigma.fo=~1, nu.fo=~x, family=SICHEL,
            n.cyc=60, trace=FALSE))
```

```
## [1] 609.7268
```

The fitted parameters of the "best" Sichel model are shown below. They are obtained by refitting the model using this time an ordinary quadratic polynomial in `x` for $\log(\mu)$ model rather that the orthogonal quadratic polynomial produced by `poly(x,2)`:

Figure 14.2

```
mSI<- gamlss(fish~x+I(x^2), sigma.fo=~1, nu.fo=~x, data=species, family=SICHEL,
             n.cyc=60)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 613.7
## GAMLSS-RS iteration 2: Global Deviance = 602.7
## GAMLSS-RS iteration 3: Global Deviance = 598.3
## GAMLSS-RS iteration 4: Global Deviance = 597.8
## GAMLSS-RS iteration 5: Global Deviance = 597.7
## GAMLSS-RS iteration 6: Global Deviance = 597.7
## GAMLSS-RS iteration 7: Global Deviance = 597.7
```

```
summary(mSI)
```

```
## ******************************************************************
## Family:  c("SICHEL", "Sichel")
##
## Call:
## gamlss(formula = fish ~ x + I(x^2), sigma.formula = ~1, nu.formula = ~x,
##     family = SICHEL, data = species, n.cyc = 60)
##
## Fitting method: RS()
##
## ------------------------------------------------------------------
## Mu link function:  log
## Mu Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.78820    0.17161   16.25   <2e-16 ***
## x           -0.00638    0.06687   -0.10    0.924
## I(x^2)       0.01396    0.00550    2.54    0.014 *
```

```
## ---
## 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)    0.367      0.463    0.79     0.43
##
## ------------------------------------------------------------------
## Nu link function:  identity
## Nu Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -11.501      3.111   -3.70  0.00045 ***
## x              1.141      0.325    3.51  0.00082 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## ------------------------------------------------------------------
## No. of observations in the fit:  70
## Degrees of Freedom for the fit:  6
##       Residual Deg. of Freedom:  64
##                       at cycle:  7
##
## Global Deviance:     597.7
##             AIC:     609.7
##             SBC:     623.2
## ******************************************************************
```

```
plot(fish~log(lake), data=species)
lines(species$x[order(species$lake)],fitted(mSI)[order(species$lake)],
      col="red")
```

The fitted model μ together with the data are shown in Figure 14.2. Figures 14.3(a) and 14.3(b) give the fitted distribution of the number of fish species for observation 40, with lake area of 165 and $(\hat{\mu}, \hat{\sigma}, \hat{\nu}) = (22.64, 1.44, -5.68)$, and observation 67, with lake area 8264 and $(\hat{\mu}, \hat{\sigma}, \hat{\nu}) = (47.78, 1.44, -1.2)$, respectively.

```
pdf.plot(mSI,c(40,67), min=0, max=110, step=1)
```

Figure 14.3

Table 14.1, taken from Rigby et al. [2008] gives the global deviance (DEV), AIC and SBC for specific models fitted to the fish species data, and is used to answer the questions at the start of this section. The terms `1`, `x` and `x<2>` indicate constant, linear and quadratic terms respectively, while the term `cs(x,3)` indicates a cubic smoothing spline with three degrees of freedom on top of the linear term `x`. Table 14.1 includes additional distributions to those previously fitted.

The following analysis is from Rigby et al. [2008]. Comparing models 2, 3 and 4 indicates that a quadratic model for log μ is found to be adequate (while the linear and the cubic spline models models was found to be inappropriate here). Comparing model 1 and 3 indicates that Y has a highly overdispersed Poisson distribution. Comparing model 3 with models 5 and 6 shows

R code on page 348

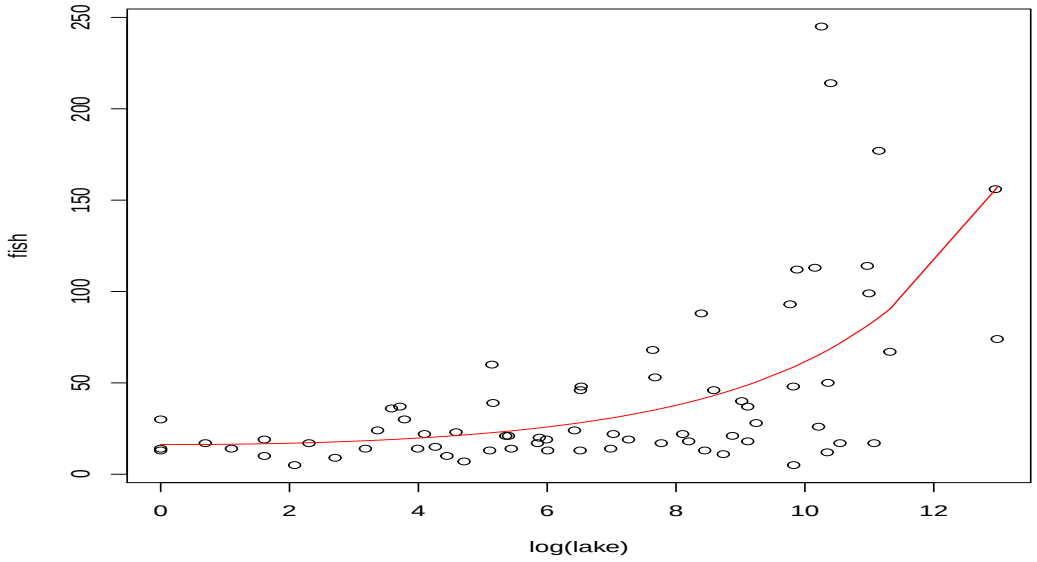


Figure 14.2: Fitted μ (the mean number of fish species) against log lake area

R code on page 349

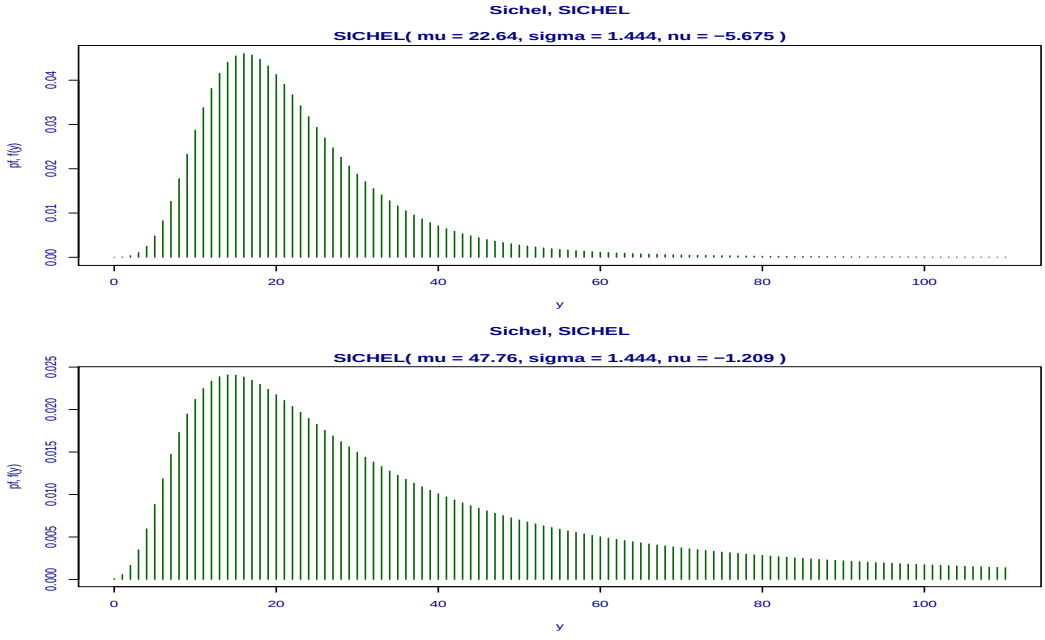


Figure 14.3: Fitted Sichel distributions for observations (a) 40 and (b) 67

that either a linear model in x for $\log(\sigma)$ or a different variance-mean relationship from that of the negative binomial (NBI) [i.e $V[Y] = \mu + \sigma\mu^2$] is required. In particular the estimated ν parameter in the negative binomial family (NBF) of model 6 is $\hat{\nu} = 2.9$ suggesting a possible variance-mean relationship $V[Y] = \mu + \sigma\mu^3$. Modelling σ in the NBF did not improve the fit greatly, as shown by model 7. A search of alternative mixed Poisson distributions included the Poisson-inverse Gaussian (PIG), the Sichel (SI) and and the Delaporte (DEL). The models with the best AIC for each distribution were recorded in Table 14.1 models 8 to 11. A normal random effect mixture distribution was fitted (using 20 Gaussian quadrature points) to the Poisson and NBI conditional distributions giving models 12 and 13, i.e. Poisson-Normal and NBI-Normal respectively. 'Non-parametric' random effects (effectively finite mixtures) (NPFM),were also fitted to Poisson and NBI conditional distributions giving models 14 and 15, i.e. PO-NPFM(6) and NB-NPFM(2) with 6 and 2 components respectively. Efron's double exponential (Poisson) distribution was fitted giving model 16 (DPO). The best discretized continuous distribution fitted was a discrete inverse Gaussian distribution giving model 17 (IGdisc), again suggesting a possible cubic variance-mean relationship.

Overall the best model according to Akaike information criterion (AIC) is model 9, the Sichel model, following closely by model 11, a Delaporte model. According to the Schwarz Bayesian criterion (SBC) the best model is model 17, the discetized inverse Gaussian distribution, again followed closely by model 11.

The model in Table 14.1 with the minimum AIC value 609.7 was selected, i.e. model 9, a Sichel, $SICHEL(\mu, \sigma, \nu)$, model fitted earlier in this section, with $\log \hat{\mu} = 2.790 - 0.00638x + 0.0140x^2$, $\log \hat{\sigma} = 0.367$ and $\hat{\nu} = -11 + 1.048x$. For comparison model 11 gives the Delaporte, $DEL(\mu, \sigma.\nu)$, model (with lowest AIC). Note in model 11 that $\sigma = 1$ is fixed in the Delaporte distribution, corresponding to a Poisson-shifted exponential distribution, giving fitted model $\log \hat{\mu} = 2.787 - 0.004207x + 0.013959x^2$, $\sigma = 1$ (fixed) and logit $\hat{\nu} = 1.066 - 0.2854x$.

The following code can be used to reproduce the results of Table 14.1.

```
library(gamlss.mx)
 m1 <- gamlss(fish~poly(x,2), data=species, family=PO, trace=FALSE)
 m2 <- gamlss(fish~x, data=species, family=NBI, trace=FALSE)
 m3 <- gamlss(fish~poly(x,2), data=species, family=NBI, trace=FALSE)
 m4 <- gamlss(fish~cs(x,3), data=species, family=NBI, trace=FALSE)
 m5 <- gamlss(fish~poly(x,2), sigma.fo=~x, data=species, family=NBI,
              trace=FALSE)
source('/Users/stasinom/Dropbox/gamlss/R-code/NBFamily/NBF.r')# take it out
 m6 <- gamlss(fish~poly(x,2), sigma.fo=~1, data=species, family=NBF, n.cyc=200,
              trace=FALSE)
 m7 <- gamlss(fish~poly(x,2), sigma.fo=~x, data=species, family=NBF, n.cyc=100,
              trace=FALSE)
 m8 <- gamlss(fish~poly(x,2),  data=species, family=PIG, trace=FALSE)
 m9 <- gamlss(fish~poly(x,2), nu.fo=~x, data=species, family=SICHEL,
              trace=FALSE)
 m10 <- gamlss(fish~poly(x,2), nu.fo=~x, data=species, family=DEL, n.cyc=50,
               trace=FALSE)
 m11 <- gamlss(fish~poly(x,2), nu.fo=~x, data=species, family=DEL,
               sigma.fix=TRUE, sigma.start=1, n.cyc=50, trace=FALSE)
 m12 <- gamlssNP(fish~poly(x,2), data=species, mixture = "gq", K=20,
```

```
                family=PO, trace=FALSE)
m13 <- gamlssNP(fish~poly(x,2), sigma.fo=~x, data=species, mixture = "gq",
                K=20, family=NBI, trace=FALSE)
m14 <- gamlssNP(fish~poly(x,2), data=species, mixture = "np", K=6, family=PO,
                trace=FALSE)
m15 <- gamlssNP(fish~poly(x,2), data=species, mixture = "np", K=2, family=NBI,
                trace=FALSE)
m16 <- gamlss(fish~poly(x,2), nu.fo=~x, data=species, family=DPO,
              trace=FALSE)
library(gamlss.cens)
m17 <- gamlss(Surv(fish,fish+1,type= "interval2")~x+I(x^2), sigma.fo=~1,
              data=species, family=cens(IG, type="interval"))
```

```
## GAMLSS-RS iteration 1: Global Deviance = 603.2793
## GAMLSS-RS iteration 2: Global Deviance = 603.2793
```

```
GAIC(m1, m2, m3, m4, m5, m6, m7, m8, m9, m10, m11, m12, m13, m14, m15, m17)
```

```
##           df       AIC
## m9   6.00000  609.7268
## m11  5.00000  610.6493
## m17  4.00000  611.2793
## m10  6.00000  612.6593
## m5   5.00000  614.9565
## m13  6.00000  615.7281
## m6   5.00000  616.0828
## m7   6.00000  616.9229
## m8   4.00000  621.3459
## m3   4.00000  622.3173
## m12  4.00000  623.2455
## m15  6.00000  623.8794
## m4   5.99924  623.9083
## m2   3.00000  625.8443
## m14 13.00000  627.9431
## m1   3.00000 1855.2965
```

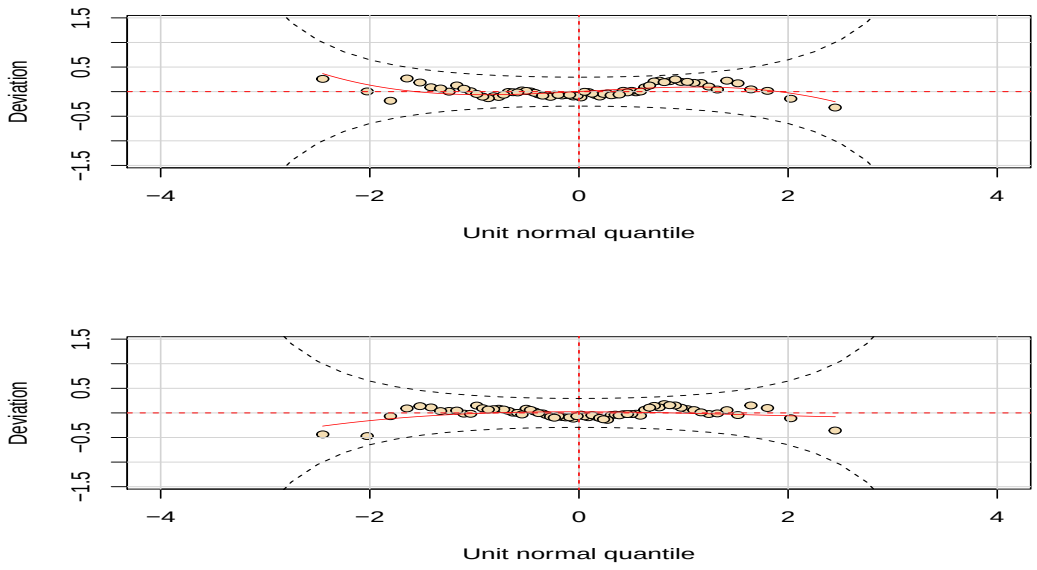
```
GAIC(m1, m2, m3, m4, m5, m6, m7, m8, m9, m10, m11, m12, m13, m14, m15, m17,
     k=log(70))
```

```
##           df       AIC
## m17  4.00000  620.2733
## m11  5.00000  621.8918
## m9   6.00000  623.2178
## m10  6.00000  626.1503
## m5   5.00000  626.1990
## m6   5.00000  627.3253
## m13  6.00000  629.2191
## m8   4.00000  630.3399
## m7   6.00000  630.4138
## m3   4.00000  631.3113
```

```
## m12  4.00000  632.2395
## m2   3.00000  632.5898
## m15  6.00000  637.3704
## m4   5.99924  637.3975
## m14 13.00000  657.1735
## m1   3.00000 1862.0420
```

```
op <- par(mfrow = c(2, 1))
wp(m9)
wp(m11)
par(op)
```

Figure 14.4



R code on page 353

Figure 14.4: Worm plots for the two 'best' model `m9` and `m17`

Worm plots for the 'best' fitted models `m9` and `m17` are shown in Figure 14.4 indicating that both models have adequate fits.

Table 14.1: Comparison of models for the fish species data

Model	$f_Y(y)$	μ	σ	ν	DEV	df	AIC	SBC
1	PO	$x<2>$	-	-	1849.3	3	1855.3	1862.0
2	NBI	x	1	-	619.8	3	625.8	632.6
3	NBI	$x<2>$	1	-	614.3	4	622.3	631.3
4	NBI	$cs(x,3)$	1	-	611.9	6	623.9	637.4
5	NBI	$x<2>$	x	-	605.0	5	615.0	626.2
6	NBI-family	$x<2>$	1	1	606.1	5	616.1	627.3
7	NBI-family	$x<2>$	x	1	604.9	6	616.9	630.4
8	PIG	$x<2>$	1	-	613.3	4	621.3	630.3
9	SICHEL	$x<2>$	1	x	597.7	6	609.7	623.2
10	DEL	$x<2>$	1	x	600.7	6	612.7	626.2
11	DEL	$x<2>$	-	x	600.6	5	610.6	621.9
12	PO-Normal	$x<2>$	1	-	615.2	4	623.2	632.2
13	NBI-Normal	$x<2>$	x	1	603.7	6	615.7	629.2
14	PO-NPFM(6)	$x<2>$	-	–	601.9	13	627.9	657.2
15	NB-NPFM(2)	$x<2>$	1	–	611.9	6	623.9	637.4
16	DPO	$x<2>$	x	-	647.3	5	655.3	664.2
17	IGdisc	$x<2>$	1	-	603.3	4	611.3	620.3

14.2 Binomial data example: the hospital stay data

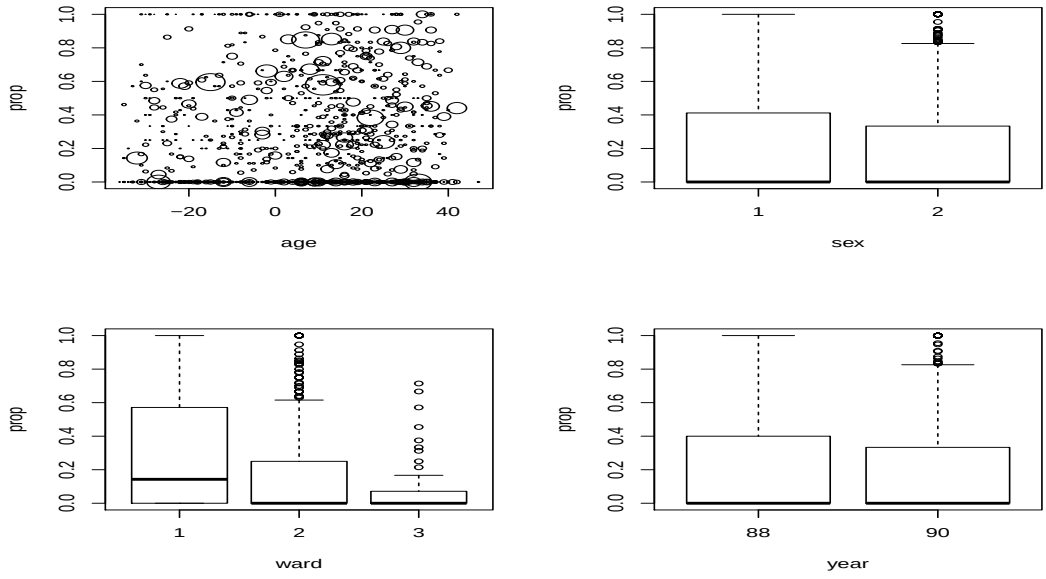
Data summary:
R data file: `aep` in package **gamlss** of dimensions 1383×8
source: Gange *et al.* (1996)
variables
`los` : total number of days
`noinap` : number of inappropriate days patient stay in hospital
`loglos` : the log of `los/10`
`sex` : the gender of patient
`ward` : type of ward in the hospital (medical, surgical or other)
`year` : 1988 or 1990
`age` : age of the patient subtracted from 55
`y` : the response variable, a matrix with columns `noinap`, `los-noinap`
purpose: to demonstrate the fitting of a beta binomial distribution to the data.
conclusion a beta binomial distribution is needed

The data, 1383 observations, are from a study at the Hospital del Mar, Barcelona during the years 1988 and 1990, see Gange *et al.* (1996). The response variable is the number of inappropriate days (`noinap`) out of the total number of days (`los`) patients spent in hospital. Each patient was assessed for inappropriate stay on each day by two physicians who used the appropriateness evaluation protocol (AEP), see Gange *et al.* (1996) and their references for more details. The following variables were used as explanatory variables, `age`, `sex`, `ward`, `year` and `loglos`.

A plot of the inappropriateness rates `ninap/los` against `age`, `sex`, `ward` and `year` are shown in Figure 14.5 obtained by:

Figure 14.5

```
data(aep)
prop<-with(aep, noinap/los)
op <- par(mfrow = c(2, 2))
plot(prop~age, data=aep, cex=los/30)
plot(prop~sex,data=aep)
plot(prop~ward,data=aep)
plot(prop~year,data=aep)
par(op)
```



R code on page 355

Figure 14.5: The rate of appropriateness against `age`, `sex`, `ward` and `year`

Gange, S. J., Munoz, A., Saez, M. and Alonso [1996] used a logistic regression model for the number of inappropriate days, with binomial and beta binomial errors and found that the later provided a better fit to the data. They modelled both the mean and the dispersion of the beta binomial distribution (BB) as functions of explanatory variables using the epidemiological package EGRET, Cytel Software Corporation (2001), which allowed them to fit a parametric model using a `logit` link for the mean and an identity link for the dispersion σ. Their final model was a beta binomial model BB(μ, σ), with terms `ward`, `year` and `loglos` in the model for `logit`(μ) and term `year` for model for σ.

First we fit their final model, equivalent to model I in Table 14.2. Although we use a log link for the dispersion σ in Table 14.2, this does not affect model I since `year` is a factor. Table 14.2 shows the GD, AIC and SBC for model I, 4519.4, 4533.4 and 4570.08 respectively. Here we are interested in whether we can improve the model using the flexibility of GAMLSS. For

the dispersion parameter model we found that the addition of `ward` improves the fit (see model II in Table 14.2 with $AIC = 4501.02$, $SBC = 4548.11$) but no other term was found to be significant. Non-linearities in the mean model for the terms `loglos` and `age` were investigated using cubic smoothing splines (`cs`), with 2 effective degrees of freedom for smoothing on top of the linear term, in models III and IV. There is strong support for including a smoothing term for `loglos` as indicated by the reduction in the AIC and SBC for model III compared to model II. The inclusion of a smoothing term for `age` is not so clear cut since, while there is some marginal support from the AIC, it is rejected strongly from SBC, when comparing model III to model IV. The R script for fitting the models in Table 14.2 is shown below:

```
mI <- gamlss(y~ward+year+loglos, sigma.fo=~year, family=BB, data=aep,
             trace=FALSE)
mII <- gamlss(y~ward+year+loglos, sigma.fo=~year+ward, family=BB, data=aep,
             trace=FALSE)
mIII <- gamlss(y~ward+year+cs(loglos,1), sigma.fo=~year+ward, family=BB,
             data=aep, trace=FALSE)
 mIV <- gamlss(y~ward+year+cs(loglos,1)+cs(age,1), sigma.fo=~year+ward,
              family=BB, data=aep, trace=FALSE)
GAIC(mI,mII,mIII,mIV, k=0) # the global deviance
```

```
##            df      AIC
## mIV  12.00010 4454.362
## mIII 10.00045 4459.427
## mII   9.00000 4483.020
## mI    7.00000 4519.441
```

```
GAIC(mI,mII,mIII,mIV) # AIC
```

```
##            df      AIC
## mIV  12.00010 4478.362
## mIII 10.00045 4479.427
## mII   9.00000 4501.020
## mI    7.00000 4533.441
```

```
GAIC(mI,mII,mIII,mIV, k=log(length(aep$age)))
```

```
##            df      AIC
## mIII 10.00045 4531.750
## mIV  12.00010 4541.147
## mII   9.00000 4548.108
## mI    7.00000 4570.065
```

Note also that the model IV can also be improved marginally by changing the logistic link for the mean to a probit link giving $GD = 4452.4$, $AIC = 4476.4$ and $SBC = 4539.1$ as shown below:

```
(mIV1 <- gamlss(y~ward+year+cs(loglos,1)+cs(age,1), sigma.fo=~year+ward,
                family=BB(mu.link="probit"), data=aep, trace=FALSE))
```

```
##
## Family:  c("BB", "Beta Binomial")
## Fitting method: RS()
##
```

Table 14.2: Models for the AEP data

Models	Links	Terms	GD (AIC) [SBC]
I	logit(μ) log(σ)	1+ward+loglos+year 1+year	4519.4 (4533.4) [4570.1]
II	logit(μ) log(σ)	1+ward+loglos+year 1+year+ward	4483.0 (4501.0) [4548.1]
III	logit(μ) log(σ)	1+ward+cs(loglos,2)+year 1+year+ward	4459.4 (4479.4) [4531.7]
IV	logit(μ) log(σ)	1+ward+cs(loglos,2)+year+cs(age,2) 1+year+ward	4454.4 (4478.4) [4541.1]

```
## Call:  gamlss(formula = y ~ ward + year + cs(loglos, 1) + cs(age, 1),
##     sigma.formula = ~year + ward, family = BB(mu.link = "probit"),
##     data = aep, trace = FALSE)
##
## Mu Coefficients:
##   (Intercept)          ward2          ward3         year90  cs(loglos, 1)
##     -0.667316      -0.244238      -0.473429       0.151170       0.240327
##    cs(age, 1)
##      0.002647
## Sigma Coefficients:
## (Intercept)       year90        ward2        ward3
##      0.2953      -0.3729      -0.7172      -1.1713
##
##  Degrees of Freedom for the fit: 12.00011 Residual Deg. of Freedom   1371
## Global Deviance:     4452.36
##             AIC:     4476.36
##             SBC:     4539.14
```

The fitted functions for all the terms for μ in model IV are shown in Figure 14.6. The fitted terms for σ are shown in Figure 14.7. They have been obtained using the function `term.plot()` as follows:

```
term.plot(mIV,  pages=1)
```

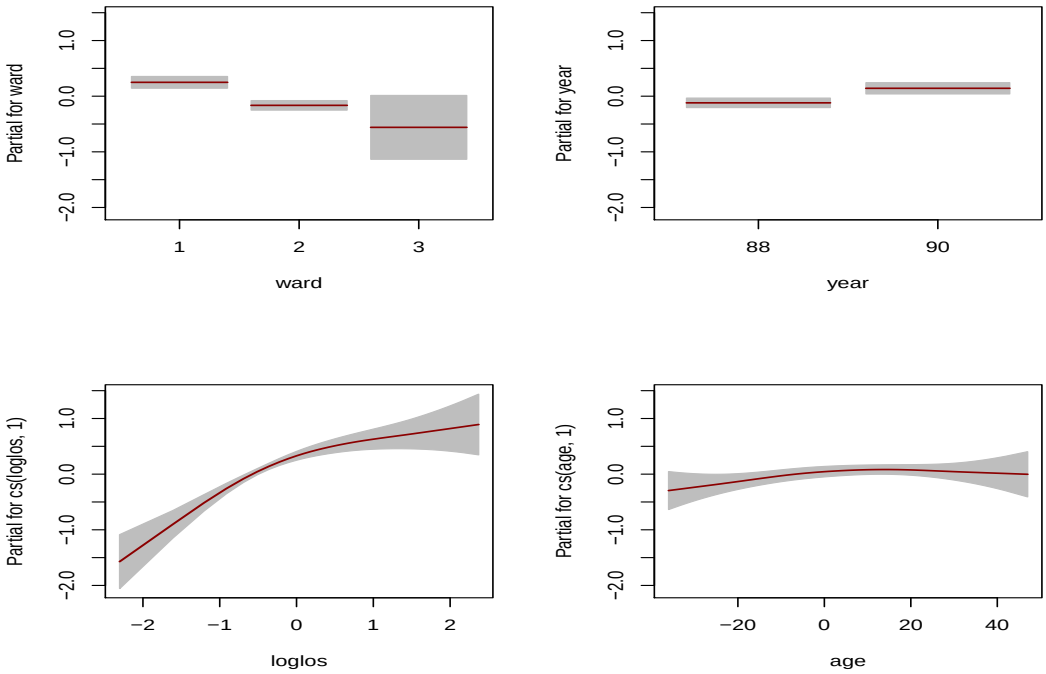
Figure 14.6

```
 term.plot(mIV, "sigma", pages=1)
```

Figure 14.7

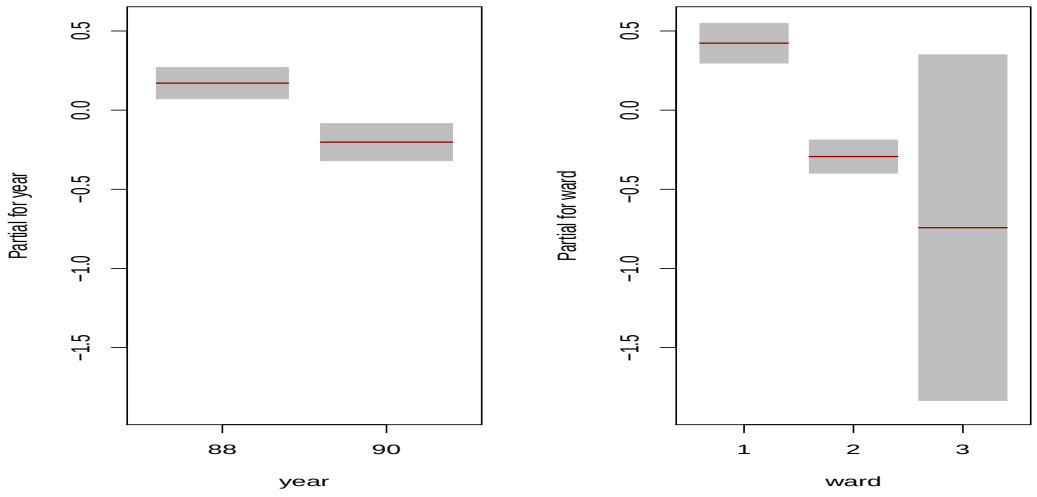
Figure 14.8 displays six instances of the normalized randomised quantile residuals (see Section **??**) from model IV. The residuals seem to be satisfactory. The figure is generated using the function `rqres.plot()`:

Figure 14.8



R code on page 357

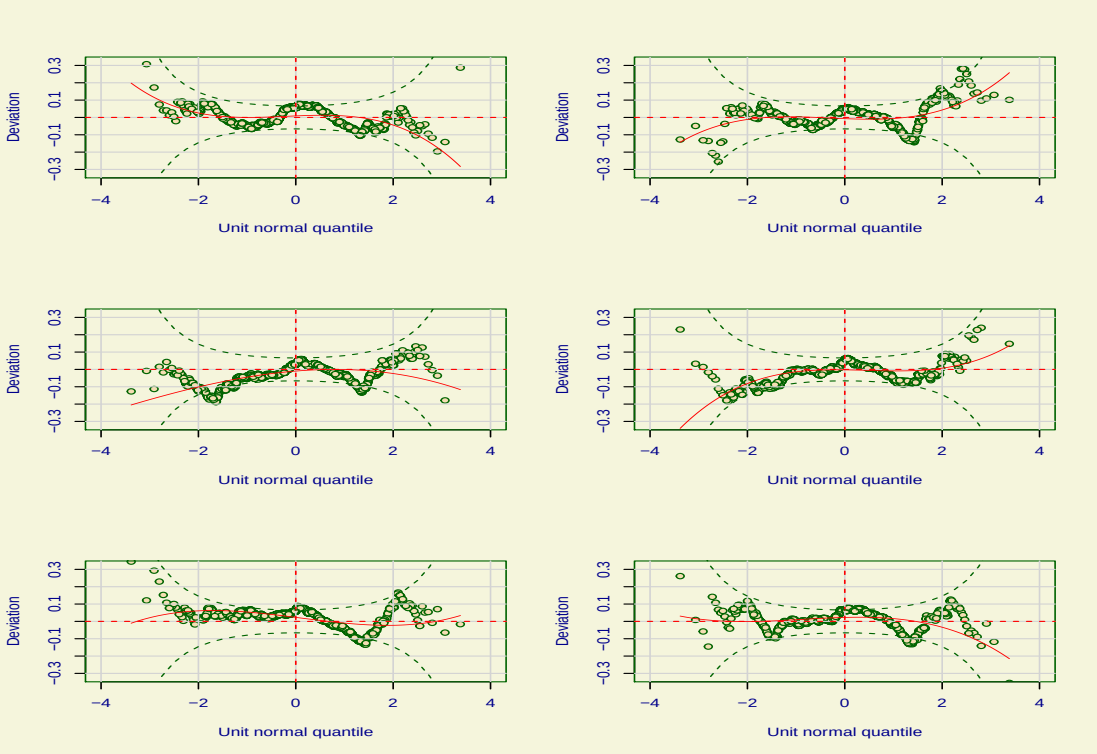
Figure 14.6: The fitted terms for μ in model IV



R code on page 357

Figure 14.7: The fitted terms for σ in model IV

```
rqres.plot(mIV)
```



R code on page 357

Figure 14.8: Six instances of the normalized randomised quantile residuals for model

14.3 Continuous distribution example: The 1990's film data

The film revenue data from the 1990's is analysed here. The data are analysed in V. Voudouris, R. Gilchrist, R. Rigby, J. Sedgwick and Stasinopoulos [2012] where more information about the data and the purpose of the original analysis can be found. We use the data here for demonstrating some of the feature of GAMLSS. The data contain several variables but here we restrict to the following:

- `lborev1` the log of box office revenues after the first week calculated in 1987 prices (the response variable)
- `lboopen` the log of box office opening revenues calculated in 1987 prices
- `lnosc` the log of the number of screens in which the film was played and
- `dist` a factor indicating whether the distributors of the film was an "Independent" or a "Major" distributor

The data can be input using the following commands:

```
library(gamlss)
data(film90)
names(film90)
```

```
##  [1] "time"        "year"        "month"       "title"       "borev0"
##  [6] "lborev0"     "nosc"        "lnosc"       "boopen"      "lboopen"
## [11] "borev1"      "lborev1"     "dist"        "whetherCost"
```

14.3.1 Preliminary analysis

Here we demonstrate how the data can be plotted in 2 and 3 dimensional plots. First we plot the response variable against the log of the number of screens and then against the log of box office opening revenues. The major and independed distributors are represented with different colours.

Figure 14.9

```
op <- par(mfrow=c(1,2))
with(film90, plot(lnosc,lborev1,pch=21,bg=c("red","green3",
    "blue","yellow")[unclass(dist)], xlab="log no of screens",
    ylab="log extra revenue"))
title("(a)")
with(film90,  plot(lboopen,lborev1,pch=21,bg=c("red","green3",
  "blue","yellow")[unclass(dist)], xlab="log opening revenue",
  ylab="log extra revenue"))
title("(b)")
par(op)
```

A good way of inspecting the data in 3-dimensions is using the package `rgl`. The following commands show how this can be done. Increase the size and rotate the figure.

```
library(rgl)
with(film90, plot3d(lboopen, lnosc, lborev1, col=c("red",
     "green3")[unclass(dist)]))
```

If you wish to show a linear least square fit to the data use the package `rpanel`:

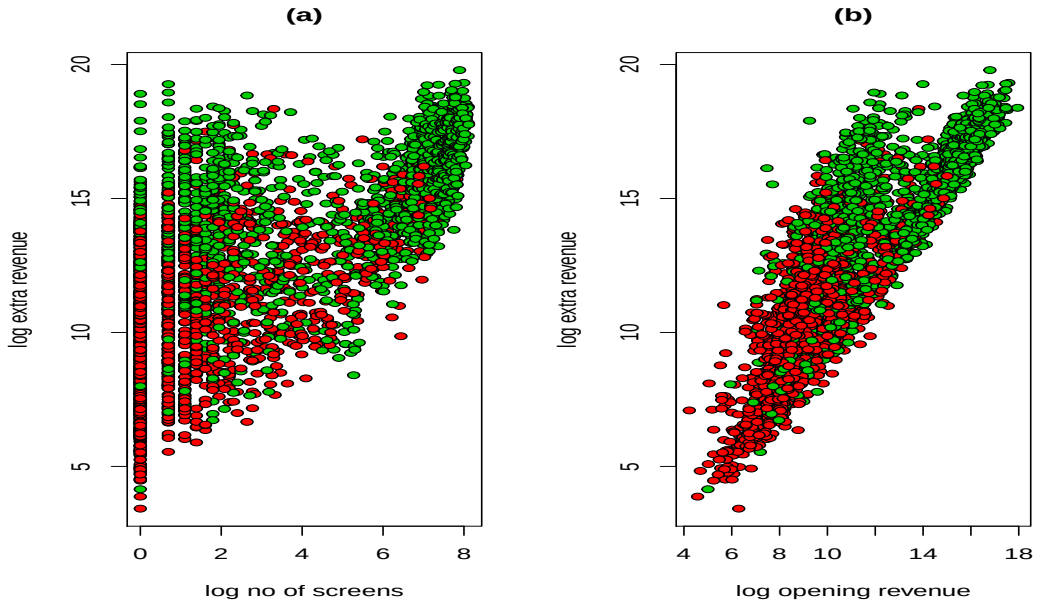
```
library(rpanel)
with(film90, rp.regression(cbind(lboopen, lnosc), lborev1))
```

You can use the option to fit a least square line in the `lboopen` direction , in the `lnosc` direction or for both vriables and also to display the residuals.

14.3.2 Modelling the data using the normal distrbution

To start the analysis we assume a normal distribution for the response variable and check whether the mean model needs:

- a linear model interaction model for `lboopen` and `lnosc`,
- an additive smoothing model for the two explanatory variables or



R code on page 360

Figure 14.9: Showing (a) `lborev1` against `lnosc` (b) `lborev1` against `lboopen`, with independent distributors represented by red color while the major distributors by green

- a fitted smooth surface

We are also checking whether we should include or exclude the factor `dist` in the mean model. Note that in order to fit a smooth surface to the data we use the function `ga()` of the package `gamlss.add` which is an interface for calling the function `gam` from Simon Wood's package `mgcv`.

```
library(gamlss.add)
# linear interaction model
 g0 <- gamlss(lborev1~lboopen*lnosc, data=film90, trace=FALSE)
g00 <- gamlss(lborev1~lboopen*lnosc+dist, data=film90, trace=FALSE)
# additive model using the pb() function
 g1 <- gamlss(lborev1~pb(lboopen) +pb(lnosc), data=film90, trace=FALSE)
 g2 <- gamlss(lborev1~pb(lboopen) +pb(lnosc)+dist, data=film90, trace=FALSE)
# fitting a surface using gam()
 g3 <- gamlss(lborev1~ga(~s(lboopen,lnosc)), data=film90, trace=FALSE)
 g4 <- gamlss(lborev1~ga(~s(lboopen,lnosc))+dist, data=film90, trace=FALSE)
GAIC(g0, g00, g1, g2, g3, g4)
```

```
##           df      AIC
## g4  27.85946 11704.65
## g3  27.32609 11762.77
## g2  18.90729 11787.63
## g1  18.59763 11866.14
```

```
## g00  6.00000 12050.10
## g0   5.00000 12194.51
```

```
GAIC(g0, g00, g1, g2, g3, g4, k=log(4031))
```

```
##           df      AIC
## g4  27.85946 11880.21
## g2  18.90729 11906.78
## g3  27.32609 11934.97
## g1  18.59763 11983.34
## g00  6.00000 12087.92
## g0   5.00000 12226.02
```

The best model seems to be model `g4` which fits a surface for `lboopen` and `lnosc` and an additive term for factor `dist`. Unfortunately a look at its residuals reveals that the normal distribution model fits very badly to the data. The following worm plot, van Buuren and Fredriks [2001], shows this clearly since most of the points lie outside the pointwise 95% confidence interval bands (shown as dashes).

Figure 14.10

```
wp(g4, ylim.all=1.1)
```

R code on page 362

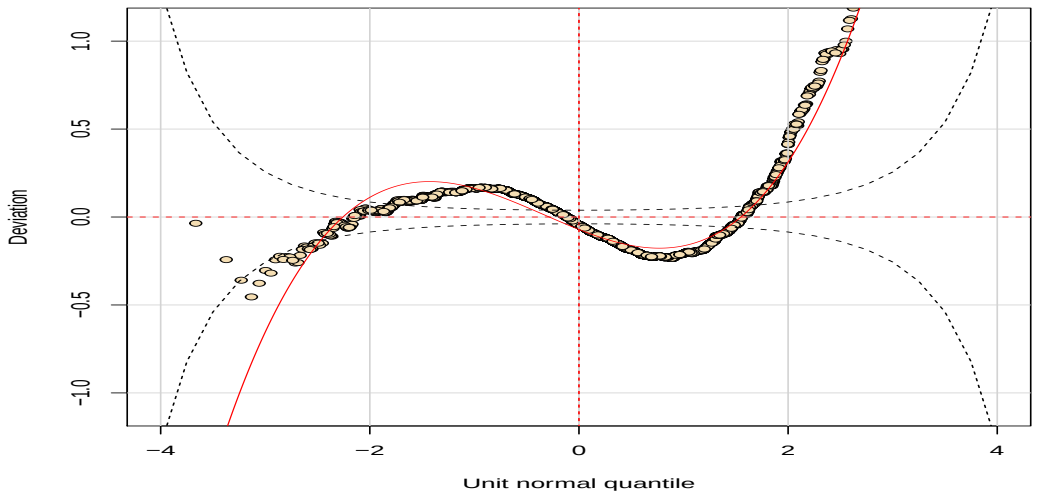


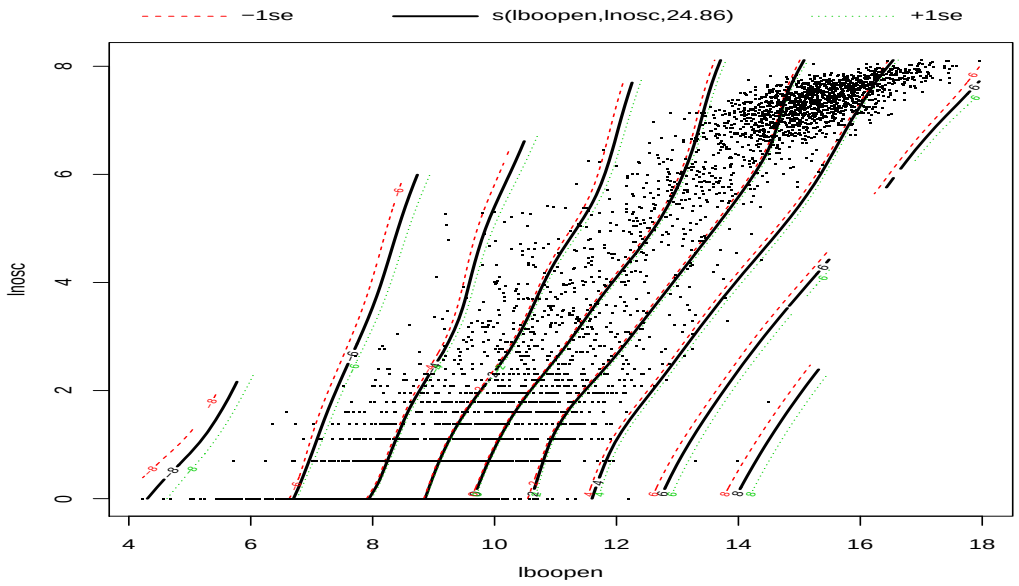
Figure 14.10: The worm plot from the normal distribution model `g4` where a fitted surfaced was used for μ

Note that in order to visualised the fitted surface you can use the `plot()` or `vis.gam()` functions of the package `mgcv`. This is because the `gam` object fitted within the backfitting algorithm is saved under the name `g4$mu.coefSmo` and can be retrieved using the function `getSmo()`.

Figure 14.11

```
plot(getSmo(g4))
```

Figure 14.12



R code on page 362

Figure 14.11: The fitted surface contour plot from model g4

```
vis.gam(getSmo(g4),theta = 0, phi = 30)
```

To check whether we need to model for σ as a function of the explanatory variables we use:

```
g42<- gamlss(lborev1~ga(~s(lboopen,lnosc))+dist,
           sigma.fo=~ga(~s(lboopen,lnosc))+dist,
           data=film90)
```

```
## GAMLSS-RS iteration 1: Global Deviance = 9859.997
## GAMLSS-RS iteration 2: Global Deviance = 9831.259
## GAMLSS-RS iteration 3: Global Deviance = 9831.384
## GAMLSS-RS iteration 4: Global Deviance = 9831.438
## GAMLSS-RS iteration 5: Global Deviance = 9831.468
## GAMLSS-RS iteration 6: Global Deviance = 9831.476
## GAMLSS-RS iteration 7: Global Deviance = 9831.48
## GAMLSS-RS iteration 8: Global Deviance = 9831.481
```

```
AIC(g42, g4)
```

```
##           df       AIC
## g42 50.89519  9933.271
## g4  27.85946 11704.648
```

```
AIC(g42, g4, k=log(4031))
```

```
##           df      AIC
```

R code on page 363

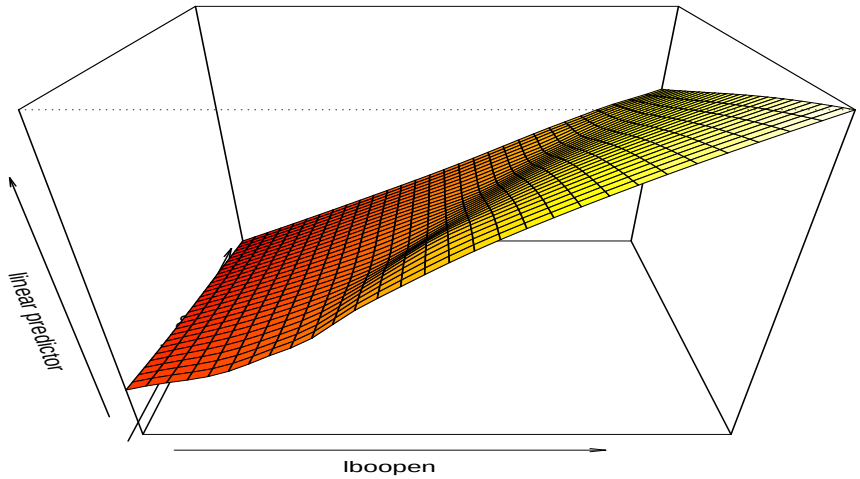


Figure 14.12: The fitted surfaced from model `g4`

```
## g42 50.89519 10254.00
## g4  27.85946 11880.21
```

Here we used a smoothing surface for `lboopen` and `lnosc` and also the factor `dist`. We found that model `g42` is superior to model `g4` whether AIC or SBC is used. To check the adequacy of the model we use a worm plot of the residuals

Figure 14.13

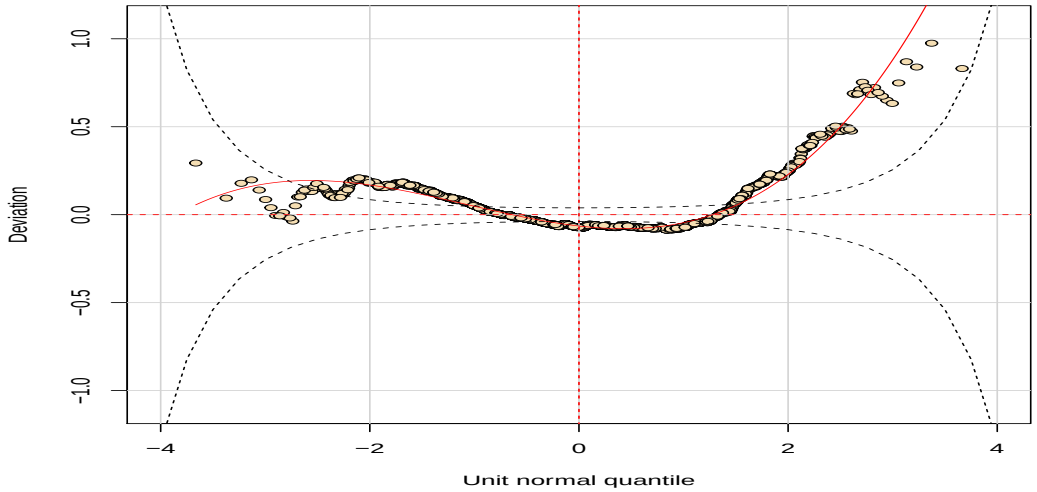
```
wp(g42, ylim.all=1.1)
```

The worm plot indicates that model `g42`, while an improvement compared to model `g4` still does not adequately explain the response variable. Next we model the response variable using the BCPE distribution.

14.3.3 Modelling the data using the BCPE distrbution

Next we model the response variable using the BCPE distribution which is a four parameter distribution defined on the positive real line. Model `mB` fits additive terms using the `pb()` function while model `mB1` uses the `ga()` function and fits smooth surfaces.

```
mB <- gamlss(lborev1 ~ pb(lboopen)+pb(lnosc) + dist,
          sigma.fo = ~ pb(lboopen)+pb(lnosc) + dist,
             nu.fo = ~ pb(lboopen)+pb(lnosc) + dist,
            tau.fo = ~ pb(lboopen)+pb(lnosc) + dist,
           family = BCPE, data = film90, n.cyc=10, trace=FALSE)
```



R code on page 364

Figure 14.13: The worm plot from the normal distribution model `g42` where a fitted surfaced was used for both μ and σ

```
mB1 <- gamlss(lborev1 ~ ga(~s(lboopen,lnosc)) + dist,
           sigma.fo = ~ ga(~s(lboopen,lnosc)) + dist,
              nu.fo = ~ ga(~s(lboopen,lnosc)) + dist,
             tau.fo = ~ ga(~s(lboopen,lnosc)) + dist,
             family = BCPE, data = film90, n.cyc=10, trace=FALSE)
AIC(g42, g4, mB, mB1)
```

```
##           df       AIC
## mB1 71.79895  9712.140
## mB  49.23464  9899.912
## g42 50.89519  9933.271
## g4  27.85946 11704.648
```

```
AIC(g42, g4, mB, mB1, k=log(4031))
```

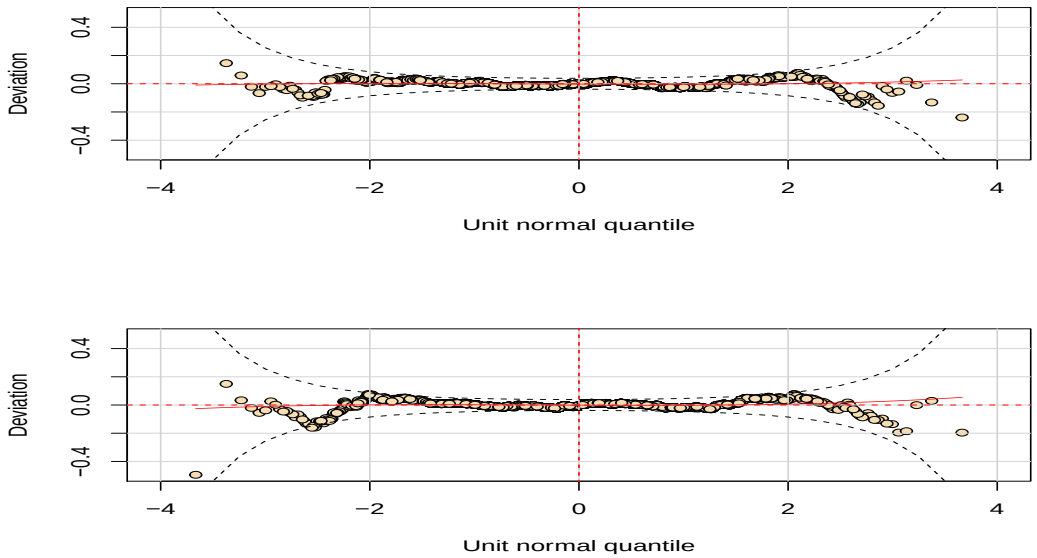
```
##           df      AIC
## mB1 71.79895 10164.60
## mB  49.23464 10210.18
## g42 50.89519 10254.00
## g4  27.85946 11880.21
```

The model `mB1` seems superior according to AIC and SBC but it uses a lot more degres of freedom. Next we plot the worm plots for both models `mB` and `mB1`.

```
op<-par(mfrow=c(2,1))
wp(mB, ylim.all=0.5)
```

Figure 14.14

```
wp(mB1, ylim.all=0.5)
par(op)
```



R code on page 365

Figure 14.14: The worm plots from the BCPE distribution models `mB` on the top and `mB1` on the botton

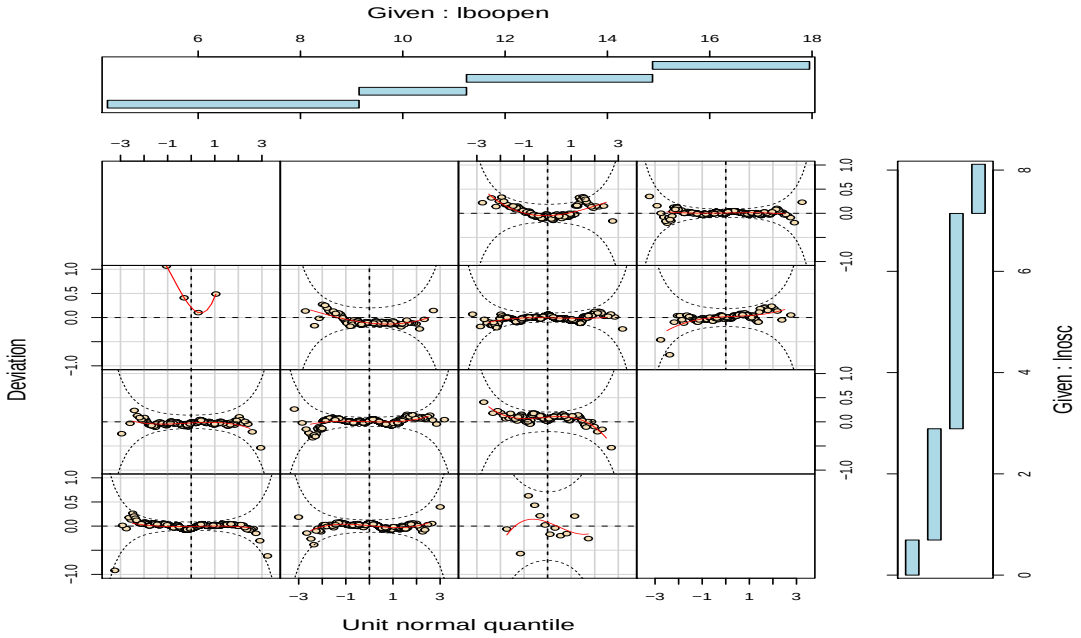
The worm plot of model (mB) (on the top) looks a bit better than model (mB1) but it is hard to decide. We can get a better idea of how the model fits in the joint ranges of the two explanatory varibles `lboopen` and `lnosc` by using a worm plot with two explanatory variables. This can be done using the following command:

Figure 14.15

```
wp(mB1, xvar=~lboopen+lnosc, ylim.worm=1)
```

```
## number of missing points from plot= 0  out of  840
## number of missing points from plot= 0  out of  377
## number of missing points from plot= 0  out of  12
## number of missing points from plot= 0  out of  315
## number of missing points from plot= 0  out of  687
## number of missing points from plot= 0  out of  147
## number of missing points from plot= 1  out of  4
## number of missing points from plot= 0  out of  151
## number of missing points from plot= 0  out of  685
## number of missing points from plot= 0  out of  174
## number of missing points from plot= 0  out of  174
## number of missing points from plot= 0  out of  834
```

In the resulting worm plot given in Figure 14.15 the 4 columns correspond to the 4 ranges of



R code on page 366

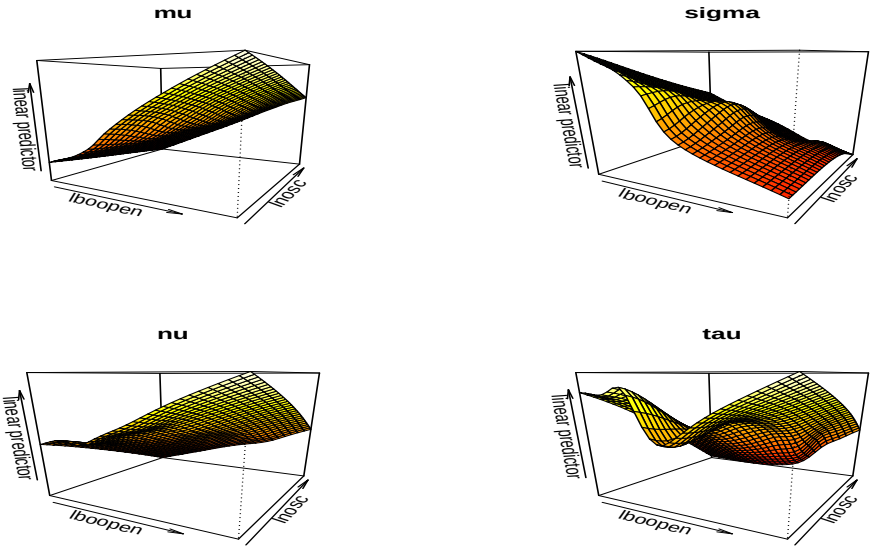
Figure 14.15: The worm plot for model `mB` for explanatory variables `lboopen` and `lnosc`

`lboopen` displayed above the plot, and the 4 rows correspond to the 4 ranges of `lnosc` displayed to the right of the plot. Within the plot there are 16 worm plots of the residuals corresponding to the 4×4 joint ranges of `lboopen` and `lnosc`. Some joint ranges have no observations within them. The worm plots generally indicate an adequate fit within the joint ranges.

The fitted smooth surfaces for μ, σ, ν and τ for model `mB1` are plotted in Figure 14.16 by using the following commands:

```
layout(matrix(c(1,2,3,4), 2, 2, byrow = TRUE))
vis.gam(getSmo(mB1,what="mu"), theta=30, phi=10)
title("mu")
vis.gam(getSmo(mB1,what="sigma"), theta=30, phi=15)
title("sigma")
vis.gam(getSmo(mB1,what="nu"), theta=30, phi=15)
title("nu")
vis.gam(getSmo(mB1,what="tau"), theta=30, phi=15)
title("tau")
layout(1)
```

Figure 14.16



R code on page 367

Figure 14.16: The fitted smooth surfaces for μ, σ, ν and τ of model `mB1`

Bibliography

M. Aitkin. Modelling variance heterogeneity in normal regression using glim. Appl. Statist., 36:332–339, 1987.

H. Akaike. Maximun likelihood identification of gaussian autoregressive moving average models. Biometrika, 60:255–265, 1973.

H. Akaike. A new look at the statistical model identification. IEEE Transactions on Automatic Control, 19(6):716–723, 1974.

H. Akaike. Information measures and model selection. Bulletin of the International Statistical Institute, 50:277–290, 1983.

Gareth Ambler. `fracpoly()`: Fractional Polynomial Model, 1999. URL `http://lib.stat.cmu.edu/S/fracpoly`. `S-PLUS`.

Christopher M Bishop et al. Neural networks for pattern recognition. 1995.

E. Borghi, M. de Onis, C. Garza, J.E. Van den Broeck, E. A. Frongillo, L. Grummer-Strawn, S. Van Buuren, H. Pan, L. Molinari, R. Martorell, A. W. Onyango, and J. C. Martines. Construction of the world health organization child growth standards: selection of methods for attained growth curves. Statistics in Medicine, 25:247–265, 2006.

John M. Chambers and Trevor J. Hastie. Statistical Models in S. Chapman & Hall, London, 1992.

G. Claeskens and N. L. Hjort. The focused information criterion. J. Am. Statist. Ass., 98: 900–916, 2003.

W. S. Cleveland and S. J. Devlin. Robust locally-weighted regression: an approach to regression analysis by local fitting. J. Am. Statist. Ass., 83:597–610, 1988.

T. J. Cole. Fitting smoothed centile curves to reference data (with discussion). Journal of the Royal Statistical Society, Series A, 151:385–418, 1988.

T. J. Cole and P. J. Green. Smoothing reference centile curves: the lms method and penalized likelihood. Statistics in Medicine., 11:1305–1319, 1992.

TJ Cole, S. Stanojevic, J. Stocks, AL Coates, JL Hankinson, and AM Wade. Age-and size-related reference ranges: A case study of spirometry through childhood and adulthood. Statistics in Medicine, 28(5):880–898, 2009.

A. Crisp and J. Burridge. A note on nonregular likelihood functions in heteroscedastic regression models. Biometrika, 81:585–587, 1994.

CYTEL Software Corporation. EGRET for Windows. CYTEL Software Corporation, Cambridge, Massachusetts, 2001.

D'Agostino, R. B., Balanger, A. and D'Agostino Jr., R. B. A suggestion for using powerful and informative tests of normality. American Statistician, 44:316–321, 1990.

C. de Boor. A Practical Guide to Splines. Springer, New York, 1978.

Dempster, A., Laird, N. and D. Rubin. Maximum likelihood from incomplete data via em algorithm (with discussion). J. R. Statist. Soc., 39:1–38, 1977.

D. Draper. Assessment and propagation of model uncertainty (with discussion). J. R. Statist. Soc. B., 57:45–97, 1995.

P. K. Dunn and G. K. Smyth. Randomised quantile residuals. J. Comput. Graph. Statist., 5: 236–244, 1996.

P. H. C. Eilers and B. D. Marx. Flexible smoothing with b-splines and penalties (with comments and rejoinder). Statist. Sci, 11:89–121, 1996.

Paul HC Eilers and Brian D Marx. Splines, knots, and penalties. Wiley Interdisciplinary Reviews: Computational Statistics, 2(6):637–653, 2010.

P.H.C. Eilers. A perfect smoother. Analytical Chemistry, 75(14):3631–3636, 2003.

R. F. Engle. ARCH. Oxford University Press, Oxford, 1995.

Robert F Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. Econometrica: Journal of the Econometric Society, pages 987–1007, 1982.

L. Fahrmeir and G. Tutz. Multivariate Statistical Modelling Based on Generalized Linear Models, 2nd ed. Springer, New York, 2001.

L. Fahrmeir and S. Wagenpfeil. Penalized likelihood estimation and iterative kalman smoothing for non-gaussian dynamic regression models. Comp. Stat. Data Anal., 24:295–320, 1997.

Ludwig Fahrmeir, Thomas Kneib, Stefan Lang, and Brian Marx. Regression: Models, methods and applications. Springer, 2013.

Fredriks, A.M., van Buuren, S., Burgmeijer, R.J.F., Meulmeester, J.F., Beuker, R.J., Brugman, E., Roede, M.J., Verloove-Vanhorick, S.P. and J. M. Wit. Continuing positive secular change in the netherlands, 1955-1997. Pediatric Research, 47:316–323, 2000.

Fredriks, A.M., van Buuren, S., Wit, J.M. and S. P. Verloove-Vanhorick. Body index measurements in 1996-7 compared with 1980. Archives of Childhood Diseases, 82:107–112, 2000.

Gange, S. J., Munoz, A., Saez, M. and J. Alonso. Use of the beta-binomial distribution to model the effect of policy changes on appropriateness of hospital stays. Appl. Statist., 45: 371–382, 1996.

Gannoun, A., Girard, S., Cuinot, C., and Saracco J. Reference curves based on non-parametric quantile regression. Statistics in Medicine, 21:3119ñ3135, 2002.

P. J. Green and B. W. Silverman. Nonparametric Regression and Generalized Linear Models. Chapman and Hall, London, 1994.

A. C. Harvey. Estimating regression models with multiplicative heteroscedasticity. Econometrica, 41:461–465, 1976.

T. J. Hastie and R. J. Tibshirani. Generalized Additive Models. Chapman and Hall, London, 1990.

T. J. Hastie and R. J. Tibshirani. Varying coefficient models (with discussion). J. R. Statist. Soc. B., 55:757–796, 1993.

Trevor Hastie. **gam**: Generalized Additive Models, 2006. URL `http://www.R-project.org`. **R** package version 0.98.

X. He. Quantile curves without crossing. The American Statistician, 51:186ñ192, 1997.

X. He and P. Ng. Cobs: Qualitative constained smoothhing via linear programming. Computational Statistics, 14:315–337, 1999.

P.J. Heagerty and M.S. Pepe. Semiparametric estimation of regression quantiles with applications. Applied Statistics, 48:533–551, 1999.

N. L. Hjort and G. Claeskens. Frequentist model average estimation. J. Am. Statist. Ass., 98: 879–899, 2003.

Peter J Huber. The behavior of maximum likelihood estimates under nonstandard conditions. In Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, volume 1, pages 221–233, 1967.

R. Koenker. Quantile regression. Cambridge University Press, Cambridge, 2005.

R. Koenker and G. Bassett. Regression quantiles. Econometrica, 46:33–50, 1978.

R. Koenker and P. Ng. Inequality constained quantile regreesion. Sankhya, The Indian Journal of Statistics, 67:418–440, 2005.

Kapitula LR and Bedrick EJ. Diagnostics for the exponential normal growth curve model. Statistics in Medicine, 24:95–108, 2005.

D. Madigan and A. E. Raftery. Model selection and accounting for model uncertainly in graphical models using occam's window. J. Am. Statist. Ass., 89:1535–1546, 1994.

P. McCullagh and J. A. Nelder. Generalized Linear Models, 2nd edn. Chapman and Hall, London, 1989.

J. A. Nelder and D. Pregibon. An extended quasi-likelihood function. Biometrika, 74:221–232, 1987.

J. A. Nelder and R. W. M. Wedderburn. Generalized linear models. J. R. Statist. Soc. A., 135: 370–384, 1972.

P. Np and Maechler M. A fast and efficient implementation on qualitatively constrained quantile smoothing splines. Statistical Modelling, 7:315–328, 2007.

Philip H. Quanjer, Sanja Stanojevic, Tim J. Cole, Xaver Baur, Graham L. Hall, Bruce H. Culver, Paul L. Enright, John L. Hankinson, Mary S. Ip, Jinping Zheng, Janet Stocks, and ERS Global Lung Function Initiative. Multi-ethnic reference values for spirometry for the 3-95-yr age range: the global lung function 2012 equations. The European respiratory journal, 40(6):1324–1343, December 2012. ISSN 1399-3003. URL `http://view.ncbi.nlm.nih.gov/pubmed/22743675`.

A. E. Raftery. Approximate bayes factors and accounting for model uncertainty in generalised linear models. Biometrika, 83:251–266, 1996.

A. E. Raftery. Bayes factors and bic, comment on 'a critique of the bayesian information criterion for model selection'. Sociological Methods & Research, 27:411–427, 1999.

R. A. Rigby and D. M. Stasinopoulos. A semi-parametric additive model for variance heterogeneity. Statist. Comput., 6:57–65, 1996a.

R. A. Rigby and D. M. Stasinopoulos. Mean and dispersion additive models. In W. Hardle and M. G. Schimek, editors, Statistical Theory and Computational Aspects of Smoothing, pages 215–230. Physica, Heidelberg, 1996b.

R. A. Rigby and D. M. Stasinopoulos. Smooth centile curves for skew and kurtotic data modelled using the Box-Cox power exponential distribution. Statistics in Medicine, 23:3053–3076, 2004.

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

R. A. Rigby and D. M. Stasinopoulos. Using the Box-Cox t distribution in gamlss to model skewness and kurtosis. Statistical Modelling, 6:209–229., 2006a.

R.A. Rigby and D.M. Stasinopoulos. Using the Box-Cox t distribution in GAMLSS to model skewness and kurtosis. Statistical Modelling, 6(3):209, 2006b. ISSN 1471-082X.

RA Rigby, DM Stasinopoulos, and C. Akantziliotou. A framework for modelling overdispersed count data, including the Poisson-shifted generalized inverse Gaussian distribution. Computational Statistics & Data Analysis, 53(2):381–393, 2008. ISSN 0167-9473.

Robert A Rigby and Dimitrios M Stasinopoulos. Automatic smoothing parameter selection in gamlss with an application to centile estimation. Statistical methods in medical research, 2013.

B. D. Ripley. Pattern Recognition and Neural Networks. Cambridge University Press, Cambridge, 1996.

Brian D Ripley. Statistical aspects of neural networks. Networks and chaosÑstatistical and probabilistic aspects, 50:40–123, 1993.

P. Royston and D. G. Altman. Regression using fractional polynomials of continuous covariates: parsimonious parametric modelling (with discussion). Appl. Statist., 43:429–467, 1994.

P. Royston and E. M. Wright. Goodness-of-fit statistics for age-specific reference intervals. Statistics in Medicine, 19:2943–2962, 2000.

David Ruppert, Matt P Wand, and Raymond J Carroll. Semiparametric regression. Number 12. Cambridge university press, 2003.

SAS Institute Inc. Enterprise Miner Software, Version 4. SAS Institute Inc, Cary, North Carolina, 2000.

Sabine K Schnabel and Paul HC Eilers. A location-scale model for non-crossing expectile curves. Stat, 2(1):171–183, 2013a.

Sabine K Schnabel and Paul HC Eilers. Simultaneous estimation of quantile curves using quantile sheets. AStA Advances in Statistical Analysis, 97(1):77–87, 2013b.

G. Schwarz. Estimating the dimension of a model. Ann. Statist., 6:461–464, 1978.

P. L. Smith. Splines as a useful and convenient statistical tool. Amer. Statist., 33:57–62, 1979.

G. K. Smyth. Generalized linear models with varying dispersion. J. R. Statist. Soc. B., 51: 47–60, 1989.

D. M. Stasinopoulos and R. A. Rigby. Detecting break points in generalised linear models. Comp. Stat. Data Anal., 13:461–471, 1992.

D.M. Stasinopoulos and R.A. Rigby. Generalized additive models for location scale and shape (GAMLSS) in R. Journal of Statistical Software, 23(7):1–46, 2007.

V. Voudouris, R. Gilchrist, R. Rigby, J. Sedgwick and D. Stasinopoulos. Modelling skewness and kurtosis with the bcpe density in gamlss. Journal of Applied Statistics, 39:1279–1293, 2012.

S. van Buuren. Worm plot to diagnose fit in quantile regression. Statistical Modelling, 7: 363–376, 2007.

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

William N. Venables and Brian D. Ripley. Modern Applied Statistics with S. Fourth Edition. Springer, 2002. URL `http://www.stats.ox.ac.uk/pub/MASS3/`. ISBN 0-387-98825-4.

A. P. Verbyla. Modelling variance heterogeneity: residual maximum likelihood and diagnostics. J. R. Statist. Soc. B., 55:493–508, 1993.

Vlasios Voudouris, Robert Gilchrist, Robert Rigby, John Sedgwick, and Dimitrios Stasinopoulos. Modelling skewness and kurtosis with the bcpe density in gamlss. Journal of Applied Statistics, 39(6):1279–1293, 2012.

A. M. Wade and A. E. Ades. Age-related reference ranges : Significance tests for models and confidence intervals for centiles. Statistics in Medicine, 13:2359–2367, 1994.

Yuedong Wang. Smoothing splines: methods and applications. CRC Press, 2011.

Ying Wei, Anneli Pere, Roger Koenker, and Xuming He. Quantile regression methods for reference growth charts. Statistics in medicine, 25(8):1369–1382, 2006.

Halbert White. A heteroskedasticity-consistent covariance matrix estimator and a direct test for heteroskedasticity. Econometrica: Journal of the Econometric Society, pages 817–838, 1980.

Edmund T Whittaker. On a new method of graduation. Proceedings of the Edinburgh Mathematical Society, 41:63–75, 1922.

Multicentre Growth Reference Study Group WHO. WHO Child Growth Standards: Length/height-for-age, weight-for-age, weight-for-length, weight-for-height and body mass index-for-age: Methods and development. Geneva: World Health Organization, 2006.

Multicentre Growth Reference Study Group WHO. WHO Child Growth Standards: Head circumference-for-age, arm circumference-for-age, triceps circumference-for-age and subscapular skinford-for-age: Methods and development. Geneva: World Health Organization, 2007.

Multicentre Growth Reference Study Group WHO. WHO Child Growth Standards: Growth velocity based on weight, length and head circumference: Methods and development. Geneva: World Health Organization, 2009.

G. N. Wilkinson and C. E. Rogers. Symbolic description of factorial models for analysis of variance. Appl. Statist., 22:392–399, 1973.

S.N. Wood. Generalized Additive Models. An introduction with R. Chapman and Hall, 2006.

E. M. Wright and P. Royston. A comparison of statistical methods for age-related reference intervals. J. R. Statist. Soc. A., 160(2):47–69, 1997.

Index