Introducing the DoubleRobGam library

Ilaria Prosdocimi

January 26, 2015

This document discusses the use of the `DoubleGam` and `DoubleRobGam`functions. These functions used to be available at `http://wis.kuleuven.be/stat/codes.html`. The background information on the two functions can be found in

Prosdocimi, I. (2010). Smooth and robust estimation of mean and dispersion functions in regression models. PhD thesis, KULeuven, available at
https://lirias.kuleuven.be/handle/123456789/280610

and this document is based on Chapter 6 of the thesis. The thesis itself was mostly based on the following papers:

Gijbels, I. and Prosdocimi, I. (2012). Flexible Mean and Dispersion Function Estimation in Extended Generalized Additive Models, *Communications in Statistics - Theory and Methods*, **41**, DOI: 10.1080/03610926.2012.654881

Croux, C., Gijbels, I. and Prosdocimi, I. (2012). Robust Estimation of Mean and Dispersion Functions in Extended Generalized Additive Models. *Biometrics*, **268**, 31-44. doi: 10.1111/j.1541-0420.2011.01630.x.

Gijbels I. and Prosdocimi I. (2011). Smooth estimation of mean and dispersion function in extended Generalized Additive Models with application to Italian Induced Abortion data. *Journal of Applied Statistics.* **38**. DOI: 10.1080/02664763.2010.550039

Gijbels, I., Prosdocimi, I. and Claeskens, G. (2010). Nonparametric estimation of mean and dispersion functions in extended Generalized Linear Models. *Test.* DOI:10.1007/s11749-010-0187-1

In the first part of this document we present the function used to fit models for the mean and the dispersion functions presented in Gijbels and Prosdocimi (2011): the `DoubleGam` function. Robust models for mean and dispersion functions as presented in Croux *et al.* (2012) can be obtained via the `DoubleRobGam` function which is discussed in the second part of this document.

The thesis and the papers mentioned above discuss in detail how Double Robust Generlised Additive Models are placed within the Generalised Additive Models and the robust modelling frameworks. Readers non familiar with Generalised Additive Models are referred to

Wood, S. W. (2006). *Generalized Additive Models: An Introduction with R*, CRC Press.
while basic background information for robust methods can be found in:

Heritier, S., E. Cantoni, S. Copt and M.-P. Victoria-Feser (2009). *Robust Methods in Biostatistics.* Wiley, Series in Probability and Statistics

1 The DoubleGam function

The `DoubleGam` function allows the user to obtain an estimate for both the mean and the dispersion function, basically implementing the methods presented in Gijbels and Prosdocimi (2010) and Gijbels and Prosdocimi (2011).

The `DoubleGam` function can fit semi-parametric models in which some covariates enter the model parametrically and others flexibly (i.e. nonparametrically). We use the Ragweed data from the library `SemiPar` to present how to use the `DoubleGam` function. The minimal requested argument to be specified in the function is the mean regression model (`formulaM).` If only this is specified a GAM model for the mean function will be estimated. Below we fit a parametric quadratic model for the mean of the Ragweed data:

```r
library(SemiPar);data(ragweed)
ragweed<-ragweed[ragweed$year==1991,]   # day.in.seas
ragweed<-ragweed[order(ragweed$day.in.seas),] # for plotting

rag1a<-DoubleGam(formulaM=ragweed ~ day.in.seas+I(day.in.seas^2),
      data=ragweed,family="poisson",selection="none")
```

The `formulaM` argument works like the classical formula for a regression model. The `selection` argument is used to define which criterion we wish to use to select the smoothing parameter value and can take values `GCV, AIC` and `none.` The default value is `GCV,` so when we only use parametric functions in the model we should change this to `none,` since no smoothing parameter needs to be selected. The type of distribution used in order to fit the data is specified via the `family` argument, as in the standard `glm` function. At the moment the `DoubleGam` function can fit models belonging to the `poisson,` the `binomial` and the `gaussian` families, the last one being the default value.

To have a flexible fit rather than a parametric shape, the `bsp` is used in the `formulaM` argument. `bsp`builds the necessary B-splines base matrix and the appropriate penalty matrix, with a coding similar to the one proposed by Eilers and Marx (1996).

```r
rag1b <- DoubleGam(ragweed ~ bsp(day.in.seas),
                    data = ragweed, family="poisson")
```

The fitted functions can be plotted with the following command (the figure obtained is displayed in Figure 1)

```
par(mfrow=c(1,1), bty = "l", lwd=2)
plot(ragweed$day.in.seas, ragweed$ragweed ,pch=19, col=8)
lines(ragweed$day.in.seas, rag1a$fitted.values, lty=2, col=4)
lines(ragweed$day.in.seas, rag1b$fitted.values)
legend("topright", col=c(4,1), lty = c(2,1), bty = "n",
      legend = c("quadratic fit", "non-parametric fit"))
```

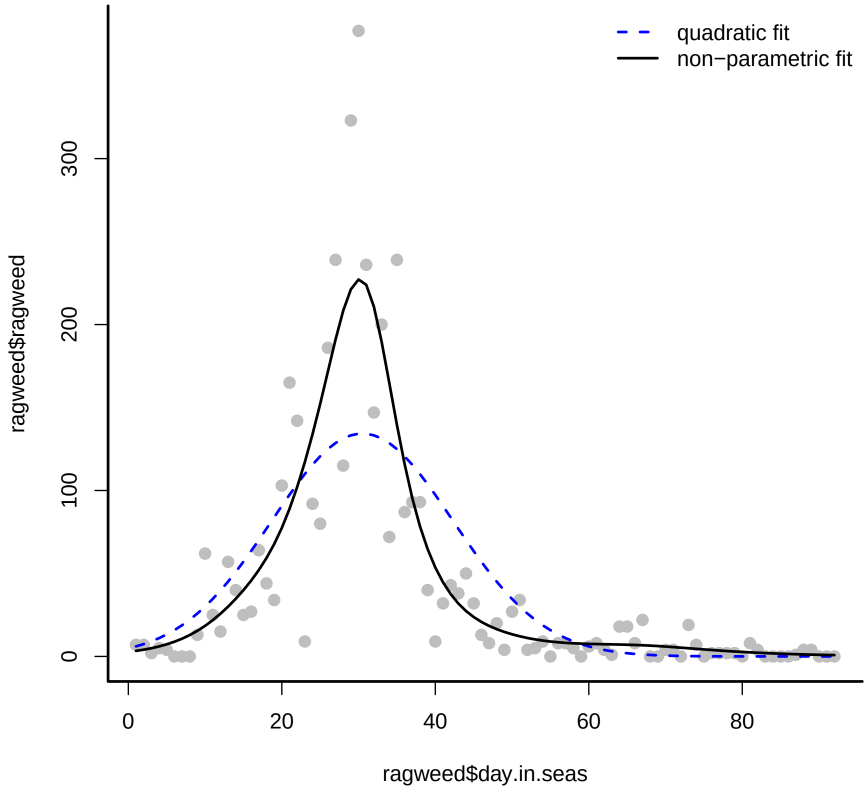

Figure 1: *The Ragweed data: mean and dispersion components.*

The `bsp` function accepts different type of arguments

```
args(bsp)

## function (x, nknots = 15, p = 3, center = TRUE, sm.p = 1, order = 2)
## NULL
```

The arguments `nknots` and `p` affect the building of the B-spline bases controlling the number of internal knots and the degree of the base. The `order` and `sm.p` arguments instead deal with characteristics of the penalization: the first defines the order m of the difference operator used in the penalization, while with `sm.p` we define the value of the smoothing parameter. If no optimal selection of the smoothing parameters is pursued, this value will be used in the fitting procedure. The `center` argument which indicates whether the B-splines base should be centered or not needs to be handled with care. The centering of the B-splines is done in order to avoid identifiability issues. In the case of a univariate covariate though, we do not need to worry about identifiability, and `DoubleGam` automatically sets `center=FALSE`. When more than one covariate is present in the model though, we recommend to not change the default value of this argument.

In order to fit both the mean and the dispersion function the `formulaG` argument also needs to be specified. The argument also follows the usual R formula fashion, although no dependent variable needs to be specified. For the Ragweed data we would have:

```
rag2<-DoubleGam(ragweed~bsp(day.in.seas),
                formulaG= ~bsp(day.in.seas,nknots=14),
                data=ragweed, family="poisson")
```

Note that when `formulaG=~1` is specified the dispersion is estimated as a constant value rather than as a function of the covariates. This is the default behaviour if no formula is specified for `formulaG` .

When estimating both the mean and the dispersion function we obtain as output two `list` objects: `fitM` and `fitG`, each one of them containing information on the fit of either the mean function $\mu(\boldsymbol{x}_d)$ or the dispersion function $\gamma(\boldsymbol{x}_d)$. Along with these two lists, the output also contains some information on the convergence of the whole algorithm. The `fitM` and `fitG` objects contain the same type of objects for the mean and the dispersion estimation.

```
names(rag2)

## [1] "fitM"      "fitG"      "convVec"   "converged" "iter"      "relE"
## [7] "data"

names(rag2$fitM)
```

```
##  [1] "coefficients"      "fitted.values"     "desMat"
##  [4] "dims"              "family"            "linear.predictors"
##  [7] "deviance"          "residuals"         "s.resid"
## [10] "y"                 "converged"         "sm.p"
## [13] "GCV"               "AIC"               "yt"
## [16] "df"                "vecdf"             "cov.coef"
## [19] "formula"           "dimsP"
```

```
names(rag2$fitG)
```

```
##  [1] "coefficients"      "fitted.values"     "desMat"
##  [4] "dims"              "family"            "linear.predictors"
##  [7] "deviance"          "residuals"         "s.resid"
## [10] "yd"                "converged"         "sm.p"
## [13] "GCV"               "AIC"               "yt"
## [16] "df"                "vecdf"             "cov.coef"
## [19] "formula"           "dimsP"
```

Most of these objects are the ones we would expect to find in a regression model output in R and can be used in the usual way. It is worth to mention that `s.resid` contains the standardized Pearson residuals and that `deviance` contains the standardized deviance residuals. `sm.p` gives information on the smoothing parameters used in the fitting, while `df` and `vecdf` give information about the total and the componentwise number of equivalent degrees freedom.

To plot the estimated mean and dispersion function simply type

```
par(mfrow=c(2,1),mai = c(0.3,0.3,0.3,0.3))
plot(ragweed$day.in.seas,rag2$fitM$fitted, type= "l")
### add the result obtained if no dispersion is estimated
lines(ragweed$day.in.seas,rag1b$fitted,lty=2, type= "l", col=4)
plot(ragweed$day.in.seas,rag2$fitG$fitted, type= "l")
```

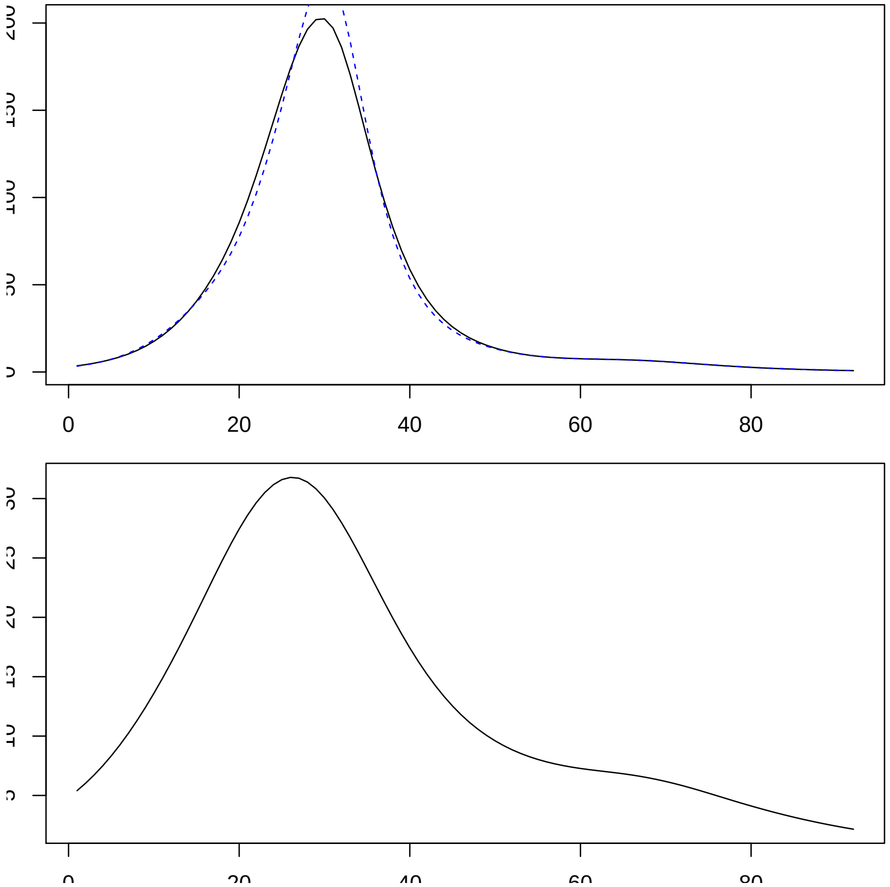

Figure 2: *The mean and dispersion function estimate .*

DoblelGam outputs belong to the S3 gamMD class, for which plot summary and print functions exist. Additive components of the estimated mean and dispersion functions can be plotted simply using

```
plot(rag2)
```

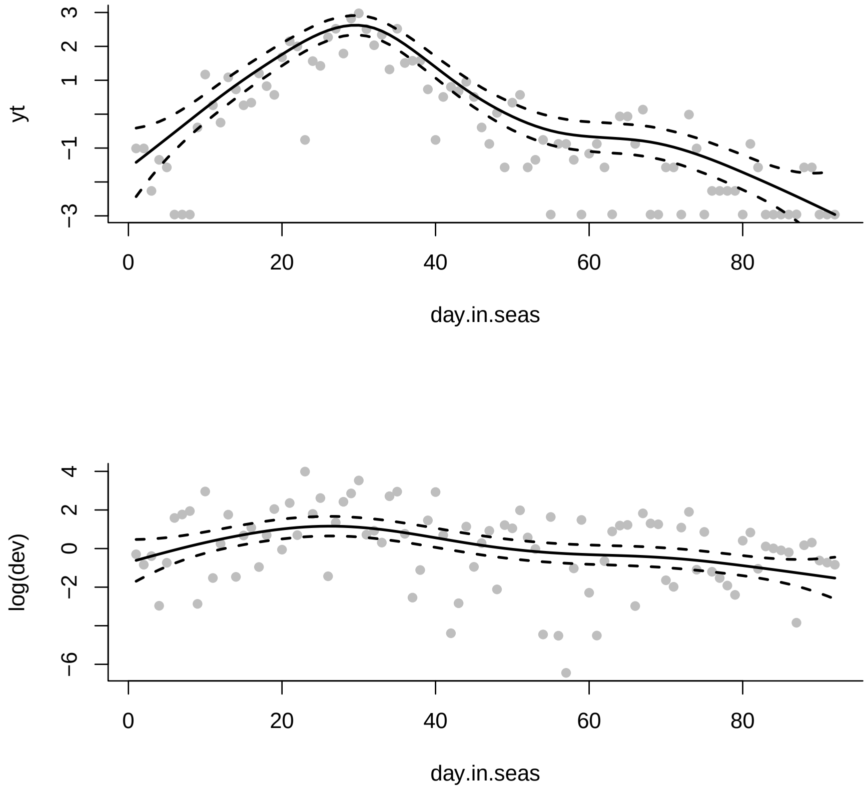

Figure 3: *The mean and dispersion functions estimate.*

The summary and print function are also available for the gamMD class.

```
class(rag2)

## [1] "gamMD"

summary(rag2)

## algorithm has converged in  3  iterations
```

```
##
##      mean estimation:
## for x1 lambdaM is 3.011 corresponding to 6.322 degrees of freedom
##      dispersion estimation:
## for x1 lambdaG is 6.643 corresponding to 3.977 degrees of freedom

rag2

## algorithm has converged in  3  iterations
##
## lambdaM is 3.011
## the e.d.f. for the mean is 7.316
##
## lambdaG is 6.643
## the e.d.f. for the dispersion is 4.977
```

The `print` function prints out information similar to the ones given by `summary`. The `plot` function plots the appropriately transformed and centered data with each of the separate mean and the dispersion components. By default the mean components are plotted above and the dispersion ones in the lower part of the graphical device (see Figure 3). If the `one` argument is set to `FALSE` two graphical devices will be displayed: one for the mean components and one for the dispersion ones.

To further illustrate the use of `DoubleGam` we use the US temperature data from the `SemiPar` library. To estimate a fit for both the mean and the dispersion of the average minimum January temperature as functions of the latitude and the longitude we use

```
library(SemiPar);data(ustemp)
temp1<-DoubleGam(formulaM=min.temp~bsp(latitude,p=2,nknots=8,sm.p=0.006)+
                          bsp(longitude,p=2,nknots=7,sm.p=0.034),
                 formulaG=~bsp(latitude,p=2,nknots=6,sm.p=1400)+
                          bsp(longitude,p=2,nknots=6,sm.p=17000),
                 selection="GCV",trace=TRUE,data=ustemp)

## The outer iteration begins
## iteration 1
## the chosen smoothing parameters are
## 0.003040804 0.05115781 for the mean - 14.05037 degrees of freedom
## 13500 26.31207 for the dispersion - 3.13126 degrees of freedom
## the relative change in the estimates is 0.01375278 0.4417377
##
## iteration 2
## the chosen smoothing parameters are
## 0.002803753 0.06211346 for the mean - 14.42481 degrees of freedom
## 13500 89.02277 for the dispersion - 3.001717 degrees of freedom
## the relative change in the estimates is 0.009648586 0.300048
##
## iteration 3
## the chosen smoothing parameters are
## 0.003402404 0.04874993 for the mean - 14.69262 degrees of freedom
## 13500 13500 for the dispersion - 3.001717 degrees of freedom
## the relative change in the estimates is 0.006038883 0.1257331
##
## iteration 4
## the chosen smoothing parameters are
## 0.003982976 0.03998633 for the mean - 14.81758 degrees of freedom
## 13500 13500 for the dispersion - 3.001717 degrees of freedom
## the relative change in the estimates is 0.002982086 0.04005848
##
## iteration 5
## the chosen smoothing parameters are
## 0.004331957 0.0367228 for the mean - 14.87032 degrees of freedom
## 13500 13500 for the dispersion - 3.001717 degrees of freedom
## the relative change in the estimates is 0.001295255 0.01132046
##
## iteration 6
## the chosen smoothing parameters are
## 0.004495987 0.03542306 for the mean - 14.89262 degrees of freedom
## 13500 13500 for the dispersion - 3.001717 degrees of freedom
## the relative change in the estimates is 0.0005881139 0.004248026
##
```

A `control` option can be used in order to keep some key points of the estimation procedure under control via an auxiliary function `DoubleGamControl`. We can for example force the function to perform a limited number of inner iterations for the estimation of either the $\boldsymbol{\alpha}_\mu$ or the $\boldsymbol{\alpha}_\gamma$ and outer iterations for the global two steps algorithm. This can be done by setting respectively the `maxitM`, `maxitG` and `maxitOUT` options to the desired number.

```
temp1<-DoubleGam(formulaM=min.temp~bsp(latitude,nknots=9,p=2)+
                           bsp(longitude,nknots=9,p=2,sm.p=0.2),
                 formulaG=~bsp(latitude,p=2,nknots=8)+
                           bsp(longitude,p=2,nknots=8,sm.p=0.54),
                 data=ustemp,
          control=DoubleGamControl(maxitM=25,maxitG=25,maxitOUT=5))

## Warning in DoubleGam(formulaM = min.temp ~ bsp(latitude, nknots
= 9, p = 2) + :  no convergence for the general algorithm
```

Also, we can change the level of accuracy requested to obtain convergence both for the inner and the outer iterations by changing the `tol` and the `acc` option. See for example how by changing the acc option in the estimation above we do not get a warning about non convergence of the algorithm after 5 iterations:

```
temp1<-DoubleGam(formulaM=min.temp~bsp(latitude,nknots=9,p=2)+
                                   bsp(longitude,nknots=9,p=2,sm.p=0.2),
                 formulaG=~bsp(latitude,p=2,nknots=8)+
                           bsp(longitude,p=2,nknots=8,sm.p=0.54),
                 data=ustemp,
      control=DoubleGamControl(maxitM=25,maxitG=25,maxitOUT=5,acc=0.07))
```

Finally, we can force the value automatically chosen for the smoothing parameters to not exceed or to not go lower than some given limits via the lambdaM and lambdaG options:

```
temp1 <- DoubleGam(formulaM=min.temp~bsp(latitude,nknots=9,p=2)+
                                   bsp(longitude,nknots=9,p=2,sm.p=0.2),
                 formulaG=~bsp(latitude,p=2,nknots=8)+
                           bsp(longitude,p=2,nknots=8,sm.p=0.54),
                 data=ustemp,
         control=DoubleGamControl(lambdaM=c(.05,200),
                                  lambdaG=c(.3,14),maxitOUT=4))

## Warning in DoubleGam(formulaM = min.temp ~ bsp(latitude, nknots
= 9, p = 2) + :  no convergence for the general algorithm
```

Changing one or more of the parameters in the DoubleGamControl function can affect quite significantly the final result. Even if for particular data analysis it can be necessary to change some of the default values to avoid convergence issues or to make the procedure faster, the control option should be used with extra care.

The default values for the DoubleGamControl function are:

```
args(DoubleGamControl)

## function (maxitM = 30, maxitG = 30, tol = 10^-5, acc = 5 * 10^-3,
##     maxitOUT = 55, lambdaM = c(1e-04, 35500), lambdaG = c(1e-04,
##         13500))
## NULL
```

2 The DoubleRobGam function

The DoubleRobGam function builds on the DoubleGam function and allows the user to model the mean and eventually the dispersion function as a nonparametric *and robust* function of one or more covariates, thus implementing the methods presented in Croux *et al.* (2013). In order to modify `DoubleGam` to allow also for robust modelling we relied on the functionalities implemented in `robustbase,` the standard R library for robust methods. In particular we used some of the code of the `glmrob` function which implements the parametric methods presented by Cantoni and Ronchetti (2001a).

The usage and the possible options of `DoubleRobGam` are very similar to the ones that were already presented for the `DoubleGam` function, most of the differences lie in the possibility of specifying how robust should the estimate be via the choice of the tuning constant c in the Huber function and in the choice of whether the smoothing parameter selection should be performed via a robust criterion. Possible values for the `selection` option in fact are, beside the standard 'none' 'GCV' and 'AIC', also 'RGCV' and 'RAIC' (see Croux *et al.* (2013)) with 'RGCV' as a default choice.

Robust and smooth estimates for the mean and the dispersion function for the US temperature data can be obtained via:

```
tempRob1<-DoubleRobGam(min.temp~bsp(latitude,nknots=9,p=2)+
                                bsp(longitude,nknots=9,p=2),
          formulaG=~bsp(latitude,p=2,nknots=8)+
          bsp(longitude,p=2,nknots=8),data=ustemp)
```

Since the class of models fitted via `DoubleRobGam` is also 'gamMD', `plotprint` and `summary` can be used.

```
summary(tempRob1)

## algorithm has converged in  8  iterations
##
##      mean estimation:
## for x1 lambdaM is 0.68 corresponding to 3.876 degrees of freedom
## for x2 lambdaM is 0.316 corresponding to 4.601 degrees of freedom
##      dispersion estimation:
## for x1 lambdaG is 16.031 corresponding to 1.868 degrees of freedom
## for x2 lambdaG is 16.784 corresponding to 1.917 degrees of freedom
```

```
plot(tempRob1)
```

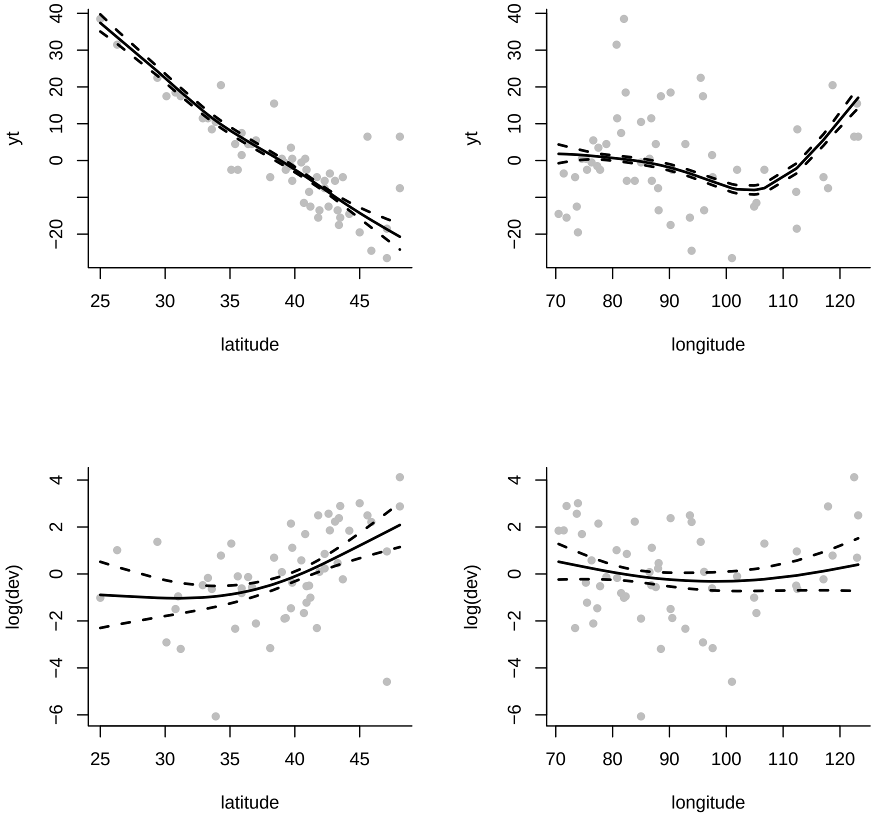

Figure 4: *Robust estimates of the mean and dispersion function estimates.*

The robustness of the estimation is governed by the tuning constant c of the Huber function $\psi_c(x)$:

$$\psi_c(x) = \begin{cases} x & \text{if } |x| \leq c \\ c \text{ sign}(x) & \text{if } |x| > c\,. \end{cases} \tag{1}$$

Lower (respectively higher) values of c correspond to more (respectively less) robust methods. The default value for c is 1.345, if one would like to have estimates which are more robust to outliers, this value should be decreased with the `control` option via the auxiliary function `DoubleRobGamControl.` This control function has, besides the same options as `DoubleGamControl,` also a `tccM` and a `tccG` option which govern the tuning constant c value for respectively the mean and the dispersion function estimation. If one wishes to fit a model with tuning constants equal to, for example, $c = 1.2$ and $c = 1.8$ for respectively the mean and the dispersion function estimation, the necessary coding would be:

```
tempRob2<-DoubleRobGam(min.temp~bsp(latitude,nknots=9,p=2)
    +bsp(longitude,nknots=9,p=2),
    formulaG=~bsp(latitude,p=2,nknots=8,sm.p=500)+
    bsp(longitude,p=2,nknots=8,sm.p=500),data=ustemp,
    control=DoubleRobGamControl(tccM=1.2,tccG=1.8))
```

The resulting fitted mean and dispersion components are depicted in Figure 5.

```
plot(tempRob2, ci.plot = FALSE)
```

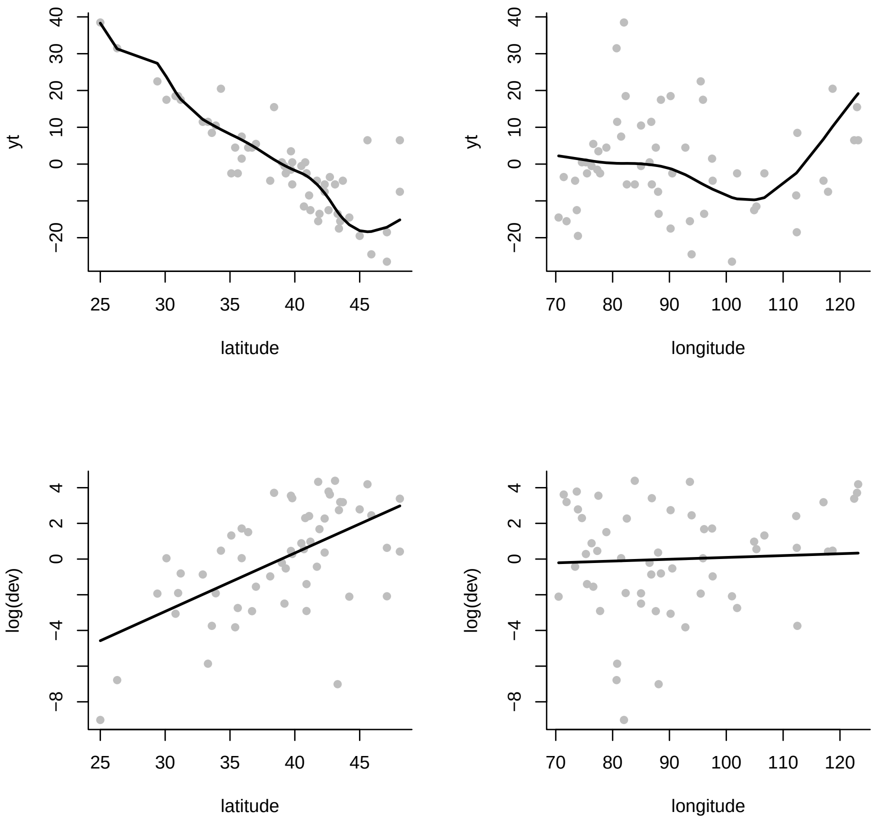

Figure 5: *The U.S. temperature data: robust fits of the mean and dispersion components as functions of the latitude and the longitude.*

Comparing Figure 4 with Figure 5 we notice that the choice of the tuning parameter seems to mostly affect the shape of the dispersion component for the latitude, while the shapes of the other components stay approximately the same. This is not always the case, and, for some data, changing the values of the tuning constants can affect very much the final fit.

One of the options that the `DoubleRobGam` function took over from `glmrob` is the `weights.on.x` option which gives the user the possibility to also use a weight function $w(\cdot)$ that can correct the estimation procedure for leverage points. The default value is `weights.on.x="none"` corresponding to $w(\cdot) = 1$, with which no action is taken against leverage points. Other possible options, mutated from the `robustbase` package are 'hat', 'robCov' and 'covMcd', which all provide different types of weights functions to correct for leverage points. We refer to `help(glmrob)` for a detailed explanation of these options.