Chapter 4
Interpretability in Generalized Additive Models

S. N. Wood, Y. Goude, and M. Fasiolo

Abstract Modelling the effect of a covariate vector on the distribution of a response variable, requires some structural assumptions, if the curse of dimensionality is to be avoided. Generalized additive models (GAMs) assume that the effects of the covariates are additive, with no, or only low order, interactions between effects. The additive assumption ensures scalability in the number of covariates and facilitates computational efficiency during model fitting. It also enhances model interpretability, which is critically important during model building and checking, as well as for communicating modelling results. This chapter formally introduces standard GAMs, as well as more flexibile GAMs for location shape and scale (GAMLSS). Is also shows how to interactively build and improve GAM and GAMLSS models via the mgcv and mgcViz R packages, which exploit their modular and interpretable structure. The final part of the chapter shows how to exploit the additive structure of GAMs to build powerful predictive models, by using random forests and online aggregation methods.

4.1 GAMs: A Basic Framework for Flexible Interpretable Regression

Suppose that we observe response variable y_i along side predictor variable vector $\mathbf{x}_i$, where $i = 1, \ldots, n$, and that the aim is to learn how to predict future y values

S. N. Wood
University of Edinburgh, School of Mathematics, Edinburgh, UK
e-mail: simon.wood@ed.ac.uk

Y. Goude
EDF R&D, London, UK
e-mail: yannig.goude@edf.fr

M. Fasiolo (✉)
University of Bristol, School of Mathematics, Bristol, UK
e-mail: matteo.fasiolo@bristol.ac.uk

A. Lepore et al. (eds.), *Interpretability for Industry 4.0 : Statistical and Machine Learning Approaches*, https://doi.org/10.1007/978-3-031-12402-0_4

from future observed $\mathbf{x}$ vectors. Note that in cases where i indexes time, $\mathbf{x}_i$ might include past values of y_i. In addition to predicting we might also be interested in understanding *how* y is related to $\mathbf{x}$—to be able to *interpret* how the prediction process works. This can be important for sanity checking predictions, and often for guiding us as to what other predictors we might usefully incorporate in $\mathbf{x}$ (if yesterday's temperature was very important, would the day before's temperature also be worth considering, for example?).

Generically we want to learn the function f in the model

$$y_i = f(\mathbf{x}_i) + \text{'noise'}$$

or more generally the probability density function, π, in

$$y_i \sim \pi(y_i|\mathbf{x}_i).$$

Unless p, the dimension of $\mathbf{x}$, is very low, then this problem suffers from the *curse of dimensionality*. For example, n^p uniformly spaced $\mathbf{x}$ points are required to cover the hypercube $[0, 1]^p$ as densely as n uniformly spaced x points could cover $[0, 1]$, and it is the density of points that determines the level of detail about f that we can hope to resolve. At the same time, even if we stick with a simple polynomial model type structure for f, we would need to use a polynomial of order at least p if we want to allow interactions involving all model terms (e.g., x_1x_2 for $p = 2$, $x_1x_2x_3$ for $p = 3$, etc.). Such a model would have at least $(2p)!/(p!p!)$ parameters to estimate. To make progress we need to impose a more restrictive structure on the problem. Additive smooth models do this by allowing a flexible smooth dependence of y on each predictor, x_j, but assume additivity of effects, with no, or only low order, interactions between effects [16, 32]. For example if $\mathbf{A}$ is a model matrix, $\mathbf{A}_i$ is its i-th row and f_j are unknown smooth functions then a *generalized additive model* has the structure

$$g(\mu_i) = \mathbf{A}_i\gamma + \sum_j f_j(x_{ji}) \qquad y_i \sim \text{EF}(\mu_i, \phi)$$

where g is a known smooth monotonic *link function* (identity, log, square root, etc.), EF denotes some exponential family distribution (Gaussian, Poisson, Gamma, bimomial etc.), $\boldsymbol{\gamma}$ is a parameter vector, ϕ a scale parameter and the f_j are subject to sum-to-zero identifiability constraints. The f_j need not be univariate: terms like $f_j(x_{ji}, x_{ki})$, $f_j(x_{ji}, x_{ki}, x_{li})$ etc. can be incorporated just as easily.

The model has a structured and interpretable form: functions of one, two and even three predictors are much easier to visualize than a general p dimensional function, for example. The curse of dimensionality is also much reduced by the decomposition into lower dimensional functions. The spacing of the predictor variables in p dimensional space is no longer the relevant consideration for how well we can estimate the model: the spacing of data in the low dimensional spaces relevant to each smooth term is now what matters. The price paid, of course, is the

assumption that only main effects and some low order interactions matter. If that is not reasonable, then nothing may be gained. Notice however, that by specifying only that the f_j are smooth functions, the model also acquires a flexibility not shared by strictly parametric models.

4.1.1 Flexibility Can Be Important

Flexibility in the specification of the f_j can really matter in applications. For example Flaxman et al. [12] attempted to estimate the time course of the pathogen reproductive number R at the start of the Covid-19 epidemic in the UK by fitting a simple epidemic model to daily data on deaths from Covid. R was allowed to vary using a step function, with steps at each of 4 government policy change dates, the last one being lockdown. A Bayesian prior was used for the step heights which promoted fewer larger steps in preference to more smaller steps. Inference with this simple parametric model suggested that R was almost unchanged until full UK lockdown, when its value dropped to below 1. This corresponds to the infection rate continuing to surge until the eve of lockdown, thereafter collapsing.

However if the simple parametric step function is replaced with a flexible smooth function of the type considered here, allowing the data a greater role in determining the function shape, then the results change dramatically [33]. $R < 1$ before full lockdown, and new infections are in decline days earlier, as shown in Fig. 4.1. The latter result has since been confirmed by more direct methods (randomly sampled antibody positive subjects were asked when their symptoms started), and is also consistent with the dynamics that later occurred under less restrictive measures, similar to the situation pertaining before full lockdown.

The rather dramatic effect of relaxing strong parametric assumptions in this case is related to the highly non-linear nature of the model, whereby features of the data

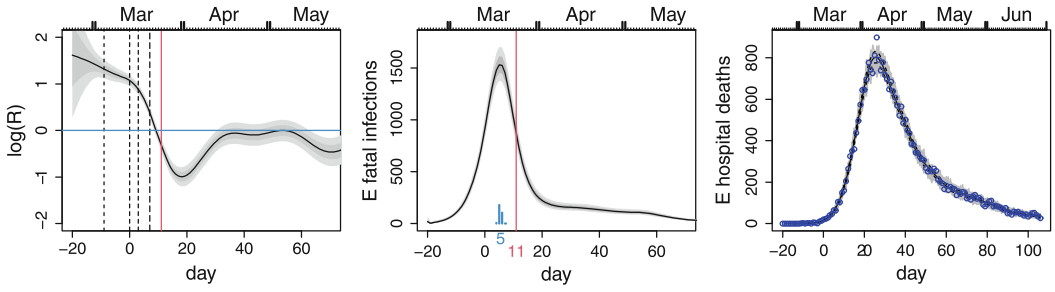


Fig. 4.1 Avoiding parameteric assumptions can make a big difference. Left: Log of inferred pathogen reproductive number, R, for England (2020), when it is assumed to be a smooth function of time: vertical lines show timing of social distancing interventions, with the red line being full lockdown. Middle: Corresponding inferred new (fatal) infections per day. Right: observed daily deaths (blue) and 100 forward simulations from inferred daily infection trajectory (grey). The original simple step function model for R in [12] implied surging infections and high R until the eve of lockdown. Subsequent data confirm the reconstruction shown here

that can not be captured with the parametric model (in particular inevitable variation in R post lockdown), result in compensating fitting artefacts that strongly effect timing. For the less non-linear model structures considered in this chapter such large effects are unlikely, but the example non-the-less serves as a warning of the potential consequences of parametric mis-specification.

4.1.2 Making the Model Computable

We need a representation of the model that can be computed with. The most fundamental components of this are a way of representing the f_j and a conventient way of measuring and controlling their smoothness. The representation part is achieved using spline type basis expansions

$$f_j(x) = \sum_{k=1}^{K} \beta_{jk} b_{jk}(x),$$

where the β_{jk} are unknown and the $b_{jk}(x)$ are simple functions chosen for nice approximation properties. K (which may vary with j, although we have not shown this notationally) is chosen to be large enough that we can be reasonably sure of avoiding the mis-specification bias that an overly restrictive model may cause. Ideally K is chosen to be somewhat larger than is likely to be necessary for representing f_j, but small enough that computation is not too onerous.

To avoid overfit with this generous choice for K we will need to control the smoothness of the f_j, by penalizing the model likelihood during estimation using wiggliness penalties, such as

$$\lambda_j \int f_j''(x)^2 dx = \lambda_j \boldsymbol{\beta}_j^{\mathsf{T}} \mathbf{S}_j \boldsymbol{\beta}_j.$$

The elements of $\mathbf{S}_j$ are known—being determined by the choice of basis functions and penalty. λ_j controls strength of penalization/smoothness of f_j—data driven selection of this parameter will be covered below. Many other choices of penalty are possible, all resulting in similar quadratic penalization of the likelihood.

A basis expansion used to fit a smooth curve, f, to data, under different degrees of penalization is shown in Fig. 4.2. A so called 'B-spline' basis is shown, but there are many other possibilities. Spline bases in general arise by mathematically working out the space of functions that minimise a particular penalty, subject to the constraint of interpolating some x, y data, or of approximating it to some specific level of accuracy. The basis shown is that for functions that minimize the penalty given above, while interpolating (or approximating) 30 x, y points, evenly spaced over the x range shown. The basis functions turn out to be made up of piecewise cubic polynomials continuous to second derivative, but having third derivative discontinuities at those 30 evenly spaced x values.

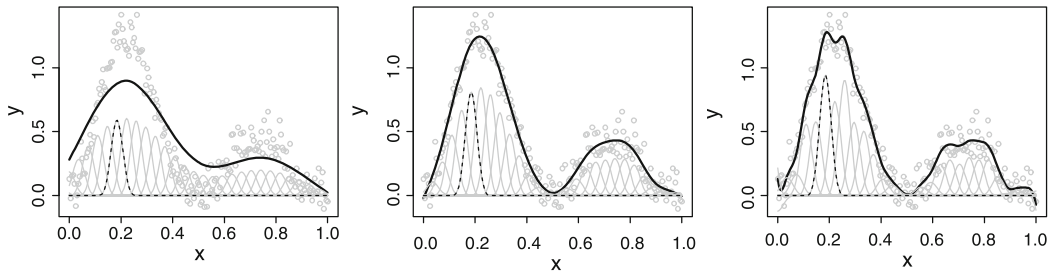


Fig. 4.2 Basis function expansions with different degrees of smoothness. Each panel shows the same set of basis functions multiplied by the corresponding coefficient in grey (i.e. $\beta_{jk} b_{jk}(x)$). The seventh is shown grey-black dashed to clearly illustrate the shape of a single basis function. The sum of the grey curves gives the black curve, which is a fit to the data shown as grey circles. The three panels show three fits, with successively higher values of the smoothing penalty, $\int f''(x)^2 dx$, allowed moving from left to right. The best fit is the middle panel, which avoids both over smoothing (left), and 'fitting the noise' (right)

$\mathbf{S}_j$ is typically not of full rank—for the given second derivative penalty example it is rank deficient by 2, the dimension of the space of functions linear in x (the functions that have zero penalty). This rank deficiency means that even with penalization, the f_j are typically not identifiable without constraint. Each f_j is only identifiable to within an intercept term. To deal with this an identifiability constraint is needed. A simple approach includes an overall intercept term in the model, but subjects each smooth to the constraint $\sum_{i=1}^{n} f_j(x_{ji}) = 0$ (or similar for functions of more than one predictor). This constraint is easily met by replacing each $b_{jk}(x)$ by

$$b^*_{jk}(x) = b_{jk}(x) - \sum_i b_{jk}(x_{ji})/n.$$

The basis now obviously spans a function space with 1 fewer degree of freedom than previously, but still with p basis functions—to remove this lack of independence we simply drop the basis function with the smallest value of $\sum_i b^*_{jk}(x_{ji})^2$ (so if the basis included the constant function—now reduced to the zero function—that's what gets dropped). Since the identifiability constraint is about removing constant functions from the basis for each smooth, the reparameterization does not change the meaning of the penalty. The constant function is in the null space of the penalty, so its removal does not alter the penalty, but of course under constraint the dimension of the penalty null space drops by one—for the penalty given above it would be one dimensional.

4.1.3 Estimation and Inference

Let us write all the GAM coefficients—$\boldsymbol{\gamma}$ and the basis coefficients for the f_j—in a single parameter vector $\boldsymbol{\beta}$. Given the exponential family distributional assumption

for y_i it is straightforward to compute the log likelihood of $\boldsymbol{\beta}$. Given choices for the smoothing parameters λ_j controlling the weight to give to smoothness in estimation, we can then find the maximum penalized likelihood estimates

$$\hat{\boldsymbol{\beta}} = \underset{\beta}{\operatorname{argmin}} \; -l(\boldsymbol{\beta}) + \boldsymbol{\beta}^{\mathsf{T}}\mathbf{S}_{\lambda}\boldsymbol{\beta}/2$$

where $l(\boldsymbol{\beta})$ is the log likelihood and where

$$\mathbf{S}_{\lambda} = \begin{bmatrix} \lambda_1\mathbf{S}_1 & 0 & \cdot \\ 0 & \lambda_2\mathbf{S}_2 & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}.$$

This penalized likelihood can be justified in its own right, but it is also convenient to view it as arising from a Bayesian approach to smooth modelling, in which the penalty is induced by an improper Gaussian *smoothing prior*

$$\pi(\boldsymbol{\beta}|\boldsymbol{\lambda}) = N(\mathbf{0}, \mathbf{S}_{\lambda}^{-}),$$

where $\mathbf{S}_{\lambda}^{-}$ is a pseudoinverse of $\mathbf{S}_{\lambda}$ (the precision matrix of the Gaussian prior). In that case $\hat{\boldsymbol{\beta}}$ is obviously the posterior mode for $\boldsymbol{\beta}$, but, in a regular large sample limit settings we also have the result that,

$$\boldsymbol{\beta}|\mathbf{y}, \boldsymbol{\lambda} \sim N(\hat{\boldsymbol{\beta}}, (\hat{\mathcal{I}} + \mathbf{S}_{\lambda})^{-1}) \tag{4.1}$$

where $\hat{\mathcal{I}}$ is the Hessian of the negative log likelihood, $-l$. This large sample Gaussian approximate posterior density will be denoted $\pi_g(\boldsymbol{\beta}|\mathbf{y}, \boldsymbol{\lambda})$ below.

Approximation (4.1) can immediately be used to find credible intervals for $\boldsymbol{\beta}$, or any quantity defined by $\boldsymbol{\beta}$, such as an f_j. But the Bayesian view offers more, since it is also possible to infer appropriate values for the smoothing parameters, $\boldsymbol{\lambda}$, in this framework. For example, we can find $\hat{\boldsymbol{\lambda}}$ to maximize the Laplace Approximate Marginal Likelihood (LAML) $\pi_g(\mathbf{y}|\boldsymbol{\lambda}) = \pi(\mathbf{y}|\hat{\boldsymbol{\beta}})\pi(\hat{\boldsymbol{\beta}}|\boldsymbol{\lambda})/\pi_g(\hat{\boldsymbol{\beta}}|\mathbf{y}, \boldsymbol{\lambda})$ (so called because the expression is exactly the first order Laplace approximation to the marginal likelihood, $\int \pi(\mathbf{y}|\boldsymbol{\beta})\pi(\boldsymbol{\beta}|\boldsymbol{\lambda})d\boldsymbol{\beta}$).

The attentive reader will notice that these Bayesian computations have the same structure as those required for a generalized linear mixed model with Gaussian random effects—the smoothing parameters playing the part of inverse variance components. This duality between smooths and random effects means that many simple Gaussian random effect terms can be included in a GAM and treated just like the f_j, statistically and computationally.

Computation of $\hat{\boldsymbol{\lambda}}$ is usually accomplished using a Newton or Quasi-Newton method, adapted to deal with the common case in which the optimal value for a λ_j is infinite, and using $\log \lambda_j$ as the working parameters, to keep λ_j positive. Computation of the LAML and its derivatives, required for optimization, involves

recomputing the $\hat{\boldsymbol{\beta}}$ corresponding to each trial $\hat{\boldsymbol{\lambda}}$: an 'inner' Newton optimization is usually employed for this. The derivatives of $\hat{\boldsymbol{\beta}}$ w.r.t. to $\log \lambda_j$ are also required in order to compute derivatives of the marginal likelihood w.r.t. $\log \lambda_j$ for the outer optimization—implicit differentiation methods are usually used. More details can be found in [34] or [32], for example.

4.1.4 Checking, Effective Degrees of Freedom and Model Selection

Model checking with an exponential family GAM proceeds much as for a GLM. Similar residual checks are made to look for violations of the modelling assumptions, particularly violation of the assumed mean-variance relationship for y, and for any un-modelled auto-correlation in the residuals. Un-modelled auto-correlation in residuals ordered by a predictor can sometimes be an indicator that the basis dimension, K, used for the corresponding smooth term, was overly restrictive and a larger value is needed. The possibility that a K value is too small is generally something to bear in mind when model checking and is additional to the usual GLM checks.

One indicator that K *may* be too small is if the *effective degrees of freedom* of a smooth term is close to $K - 1$ (the -1 resulting from the identifiability constraint). Effective degrees of freedom attempt to capture the notion of how many unpenalized coefficients would result in a model term of equivalent flexibility to a given set of penalized coefficients, estimated under a particular level of penalization.

As a motivating example consider a smooth of one predictor, estimated with a squared second derivative penalty penalizing the likelihood, and having a basis dimension of 20. Under very high penalization (high λ_j) the smooth will be constrained to lie in the null space of the penalty—i.e. to be linear in the predictor. Despite its 19 coefficients the line has only one degree of freedom, corresponding to its slope (the constraint having removed the intercept). Under no penalization the smooth would obviously have 19 degrees of freedom, one for each free parameter. As we vary its smoothing parameter from infinity down to zero, the smooth becomes continuously less smooth and more complex, and it makes sense that for a level of smoothness somewhere between the straight line case and full unpenalized flexibility, the degrees of freedom should also be intermediate.

Consider the matrix $\mathbf{F} = (\hat{\mathcal{I}} + \mathbf{S}_\lambda)^{-1}\hat{\mathcal{I}}$ evaluated at $\hat{\boldsymbol{\beta}}$. This is the matrix approximately mapping the unpenalized $\hat{\boldsymbol{\beta}}$ to the penalized version, and so its diagonal elements, F_{ii}, can be viewed as 'shrinkage' factors for the coefficients. Summing up the elements F_{ii} corresponding to a single smooth term gives its *effective degrees of freedom* (EDF). In fact, since $\hat{\mathcal{I}}$ is not guaranteed to be positive definite at $\hat{\boldsymbol{\beta}}$, it is usually better to use the expected Hessian of the negative log likelihood in the computation of $\mathbf{F}$, to avoid occasional nonsensical EDF results. With this modification the EDF always decreases monotonically with increasing

smoothing parameters and is bounded between $K - 1$ and the penalty null space dimension.

In addition to inference, such as credible interval computation, directly using the Bayesian result (4.1), some frequentist model selection tools are also useful. In particular it is possible to construct a generalized AIC

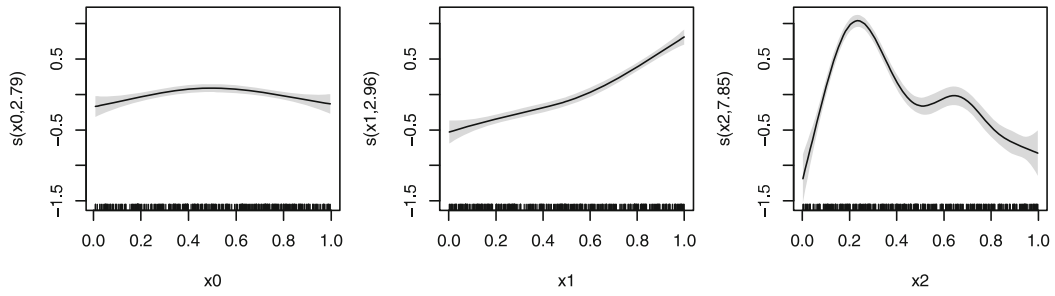
$$\text{AIC} = -2l(\hat{\boldsymbol{\beta}}) + 2\text{trace}(\mathbf{F}),$$

for model comparison, although for best performance this should incorporate a correction to the EDF (trace($\mathbf{F}$)) accounting for smoothing parameter uncertainty [34]. It is also possible to obtain approximate p-values for testing $H_0 : f_j = 0$ [30, 31]. Both approaches can provide useful guidance about which terms should be included in a GAM.

4.1.5 GAM Computation with `mgcv` in R

R package `mgcv` offers generalized additive modelling functions based on the preceding approach. In particular its `gam` function is used in a similar manner to the `glm` function for fitting generalized linear models in R. That is a response variable and linear predictor structure are specified using a *model formula*, while the distribution and link function are specified via a *family*, with the observations of the variables referred to in the model formula usually supplied in a *data frame*. To see this in action, consider a simple simulated Poisson example from the `mgcv` help files:

```
library(mgcv);set.seed(7)
dat <- gamSim(1,n=400,dist="poisson",scale=.2,verbose=FALSE)
b <- gam(y~s(x0)+s(x1)+s(x2),family=poisson,data=dat,method="REML")
par(mfrow=c(1,3),mar=c(4,4,1,1))
plot(b,scheme=1)
```



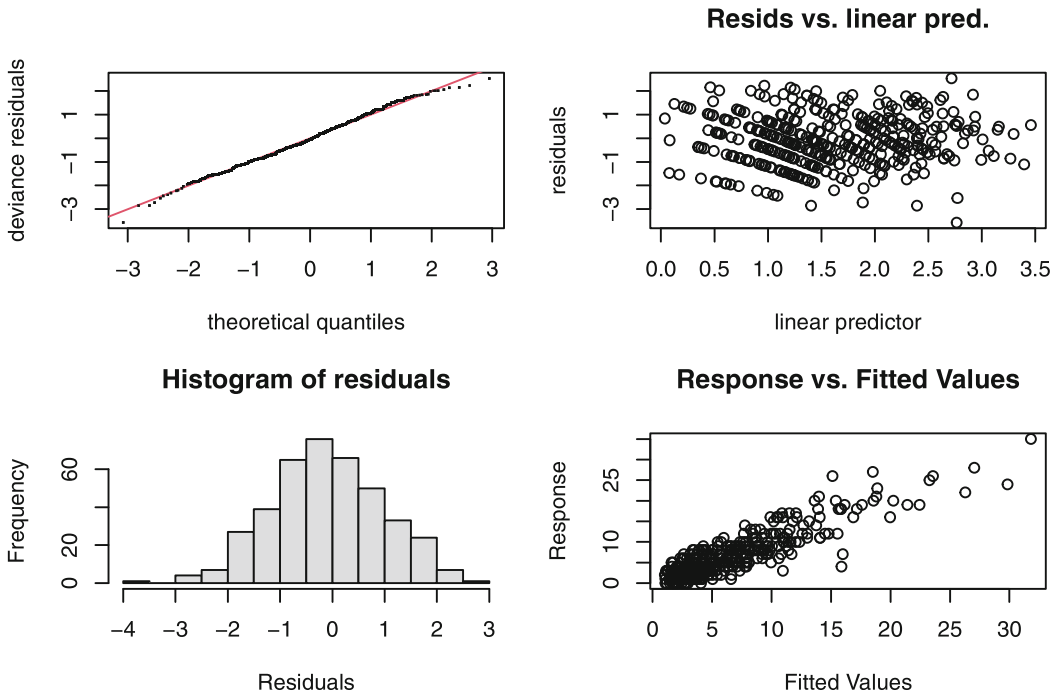
The model fitted here is

$$\log(\mu_i) = f_1(x_{0i}) + f_2(x_{1i}) + f_3(x_{2i}) \text{ where } y_i \sim \text{Poi}(\mu_i)$$

and the plot shows the three estimated smooth functions with 95% credible intervals calculated using Equation (4.1). The numbers in the y axis labels such as `s(x0,2.89)` are the termwise effective degrees of freedom, while the ticks along the x axis show where observations occurred. Default basis dimensions of 10 have been used. `s(x2,k=20)` would have changed this to 20, for example.

Before taking credible intervals or model estimates seriously it is important to do some model checking of course. Some basic residual checks are provided as follows:

```
gam.check(b)
```




```
##
## Method: REML   Optimizer: outer newton
## full convergence after 4 iterations.
## Gradient range [-2.035354e-10,5.211281e-08]
## (score 918.5614 & scale 1).
## Hessian positive definite, eigenvalue range [0.8959316,3.55858].
## Model rank =  28 / 28
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##         k'  edf k-index p-value
## s(x0) 9.00 2.79    1.03    0.76
## s(x1) 9.00 2.96    1.04    0.85
## s(x2) 9.00 7.85    1.10    0.99
```

The top right panel shows deviance residuals against the linear predictor—the residuals should show no pattern in mean or variance if the model structure is

correct and the mean-variance model is appropriate. Top left is a qq-plot of the ordered residuals against theoretical quantiles—this should be close to a straight line if the distribution model is approximately correct. The lower two panels are self explanatory. The other output provides technical information on algorithm convergence, and an informal simulation test checking for residual correlation in the residuals when ordered against each predictor, which might indicate a problem with the basis dimension being too restrictive (although other issues may also cause an apparently significant result, so this is only a rough guide). The output here shows no problems. While serious checking should involve further plotting of residuals against predictors, let us move on, and consider model selection functions such as:

```
AIC(b)

  ## [1] 1802.089

summary(b)

  ##
  ## Family: poisson
  ## Link function: log
  ##
  ## Formula:
  ## y ~ s(x0) + s(x1) + s(x2)
  ##
  ## Parametric coefficients:
  ##             Estimate Std. Error z value Pr(>|z|)
  ## (Intercept)  1.58763    0.02481   63.99   <2e-16 ***
  ## ---
  ## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
  ##
  ## Approximate significance of smooth terms:
  ##         edf Ref.df Chi.sq p-value
  ## s(x0) 2.787  3.467  14.19 0.00455 **
  ## s(x1) 2.956  3.670 404.51 < 2e-16 ***
  ## s(x2) 7.852  8.644 812.74 < 2e-16 ***
  ## ---
  ## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
  ##
  ## R-sq.(adj) =  0.785   Deviance explained = 74.1%
  ## -REML = 918.56  Scale est. = 1         n = 400
```

Most of this output is self explanatory. The p-values are each for the test of whether the corresponding model term could be zero. Another standard task is prediction using the model. Given a data frame of predictor variable values, the `predict` function provides predictions on the linear predictor scale, by model term, on the response scale, or it can produce the `"lpmatrix"` mapping the model coefficients to the required predictions. Here is an example, predicting on the linear predictor scale:

```
pd <- data.frame(x0=c(.4,.1),x1=c(.3,.8),x2=c(.7,.1))
predict(b,newdata=pd,se=TRUE)

  ## $fit
  ##        1        2
  ## 1.299005 1.964851
  ##
  ## $se.fit
  ##          1          2
  ## 0.07988181 0.07898040
```

4.1.6 Smooths of Several Predictors

Now consider smooth model terms involving more than one predictor. Interpretability of model smooth terms involving multiple predictors requires some consideration to be given to how they are constructed. There are two main possibilities. One is to construct multidimensional analogues of smooths of one predictor, for which a single smoothing penalty treats smoothness in all predictor space directions isotropically. This is saying that if we travel along any two arbitrary lines in predictor space, we would expect the function to appear more or less equally smooth along both of them. The simplest example of such a penalty is the *thin plate spline* penalty for a smooth, $f(x_j, x_k)$, of two predictors,

$$\lambda \int \frac{\partial^2 f}{\partial x_j^2}^2 + 2\frac{\partial^2 f}{\partial x_j \partial x_k}^2 + \frac{\partial^2 f}{\partial x_k^2}^2 dx_j dx_k$$

It can be generalized to smooths of any number, d, of predictors and different orders, m, of derivative, although in general functions minimizing such a penalty while interpolating a given set of points only exist if $2m > d$ ($2m > d + 1$ if f should look visually smooth). A less intuitively interpretable penalty is required to restore complete freedom in the choice of m as d increases: the Duchon spline penalty, of which the thin plate spline penalty is a special case [9].

As in the case of the univariate penalty, given a penalty the basis for the space of functions minimizing the penalty while interpolating (or approximating to given accuracy) a set of points is known mathematically, and such a thin plate spline or Duchon spline basis can be used to represent f, exactly as in the univariate case. Again, given a basis with coefficients $\boldsymbol{\beta}$, the penalty has the form $\boldsymbol{\beta}^\mathsf{T}\mathbf{S}\boldsymbol{\beta}$, where $\mathbf{S}$ is fixed and known. In `mgcv`, terms like `s(x1,x2,k=50)` are used to add isotropic thin plate spline terms to the model. More predictors can also be used, and general Duchon penalty splines are also available.

Isotropic smoothing can also be generalized to applications such as smoothing on the surface of a sphere, as illustrated in the right panel of Fig. 4.3. However isotropic

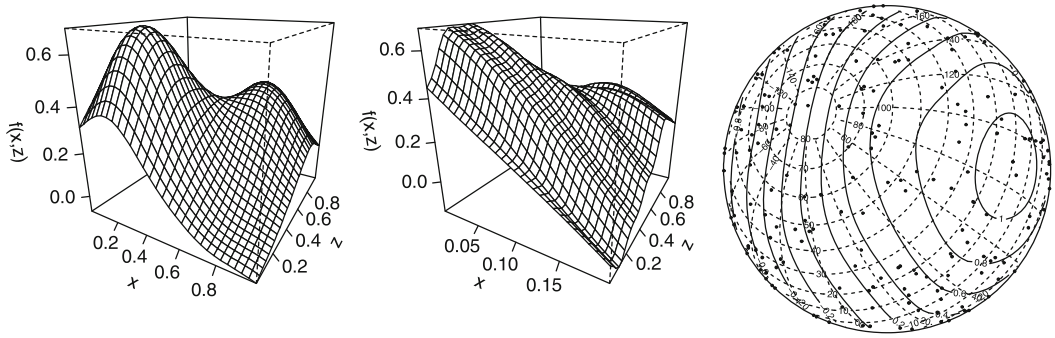


Fig. 4.3 Isotropy is a strong assumption, but isotropic smoothing is readily generalized. Left: A thin plate spline fitted to some data randomly and uniformly distributed over the unit x, z square. Middle: A thin plate spline fitted to the same data as in the left panel, but with the x co-ordinate divided by 5. The strong effect of the isotropic smoothness assumption is clearly visible: the smoother has the same level of variability per unit change in x as it has per unit change in z, but that results in a poor reconstruction here. Interaction/tensor product smooths do not suffer from this problem—they are in fact invariant to such linear rescalings. Right: an example of an isotropic smoother, where smoothness is defined over the surface of the sphere

smoothness is a strong assumption, as illustrated for a thin plate spline in the left two panels of the same figure.

The second approach is to abandon isotropy and instead use the notion of a statistical interaction to construct functions of several predictors (e.g. [29]). A statistical interaction occurs when the effect of one predictor is itself dependent on another predictor. Or more concretely, when the coefficients describing the effect of one predictor themselves depend on another predictor. Following this construction, we can construct an interaction spline of x_j and x_k by starting with a univariate spline basis expansion for the smooth of x_j,

$$f_j(x_j) = \sum_i \gamma_i g_i(x_j),$$

where the γ_i are coefficients and the g_i are basis functions. Now allow each coefficient γ_i to itself be a smooth function of x_k, using the univariate spline basis function expansion for smooths of x_k,

$$\gamma_i(x_k) = \sum_l \beta_{il} \delta_l(x_k),$$

where the β_{il} are coefficients and the δ_l are basis functions. Hence we arrive at

$$f(x_j, x_k) = \sum_i \sum_l \beta_{il} \delta_l(x_k) g_i(x_j).$$

So the basis functions for $f(x_j, x_k)$ are simply all the possible products of the basis functions for the univariate functions of x_j and x_k—for this reason the basis is often

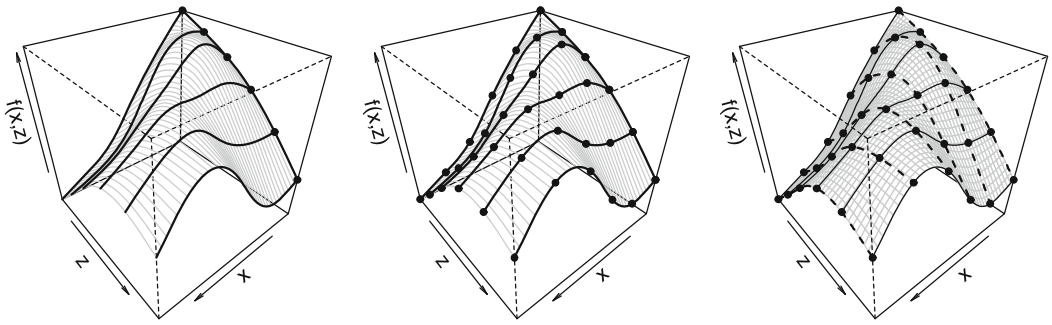


Fig. 4.4 Construction of an interaction (tensor product) smooth of z and x. Left: a smooth function of z is parameterized in terms of spline function values at 6 equally spaced z values, shown as black blobs. These parameters/coefficients are allowed to vary smoothly with x, as illustrated by the black curves, implying the surface represented by the grey curves. Middle: in practice the smooth variation of the coefficients with x is represented using spline functions of x, also parameterized in terms of function values at equally spaced x values, again shown as black blobs. Right: The construction is obviously symmetric. Smoothness of the surface $f(x, z)$ in the z direction can be measured by summing up the marginal penalties for smoothness in the z direction, for each of the dashed curves. Similarly smoothness in the x direction is measured by summing the marginal x direction penalty applied to the continuous black curves

known as a *tensor product* basis. The univariate bases used for this purpose are refered to as *marginal* bases, for obvious reasons. The construction is illustrated by Fig. 4.4, for marginal splines parameterized in terms of function values at equally spaced predictor values. Note that the construction is symmetric—the same basis arises if we start with the smooth of x_k and allow its coefficients to vary smoothly with x_j. It also generalizes immediately to more predictor variables. The coefficients of a smooth of two variables can be turned into smooth functions of a third variable, and so on.

What should we use as a smoothing penalty with such a tensor product basis? Interpretability is one important consideration. Another is that smoothing equally in all directions, as isotropic penalties do, is not really compatible with the notion of an interaction smooth. Suppose we are interested in how the effect of air temperature varies with elevation, for example. Why should we expect the same smoothness with respect to a unit change in temperature as we expect per unit change in elevation? And if we do make this choice the relative amount of smoothing with respect to elevation and temperature will be entirely dependent on the choice of measurement units, an essentially arbitrary decision.

A simple way to avoid such arbitrariness is to use a smoothing penalty that treats smoothness in different predictor variable directions separately, with at least one smoothing parameter per variable. For the example given above we might use

$$\lambda_j \int \left(\frac{\partial^2 f}{\partial x_j^2}\right)^2 dx_j dx_k + \lambda_k \int \left(\frac{\partial^2 f}{\partial x_k^2}\right)^2 dx_j dx_k. \tag{4.2}$$

With a different smoothing parameter for each predictor, the amount of smoothing appropriate w.r.t. to the two variables can be established as part of model fitting, rather than being arbitrarily fixed as part of model specification.

Computation of penalties like the one reported in Equation (4.2) is possible, but somewhat involved computationally, and a simpler alternative is equally interpretable. For any marginal basis a one-to-one linear reparameterization can be used to give the basis coefficients of the marginal smooth the interpretation of being values of the smooth at equally spaced predictor values. Hence when these coefficients are turned into smooth functions of another variable, e.g. x_k, they define smooth curves over the surface of f in the x_k direction. The whole collection of such curves can each be penalized by whatever interpretable penalty is associated with the marginal smooth for x_k. The right panel of Fig. 4.4 illustrates the idea. It then turns out that if $\boldsymbol{\gamma}^\mathsf{T}\tilde{\mathbf{S}}_j\boldsymbol{\gamma}$ was the smoothing penalty for the marginal smooth of x_j, then the x_j direction penalty for the tensor product has the form

$$\boldsymbol{\beta}^\mathsf{T}\mathbf{S}_j \otimes \mathbf{I}\boldsymbol{\beta},$$

where '$\otimes$' denotes the Kronecker product and $\mathbf{I}$ an identity matrix whose rank is the basis dimension for the x_k marginal smooth. The penalty in the x_k direction is $\boldsymbol{\beta}^\mathsf{T}\mathbf{I} \otimes \mathbf{S}_k\boldsymbol{\beta}$. Again, the construction generalized readily to more predictors.

In `mgcv` tensor product smooths using this construction are added to a model using model terms like `te(x1,x2,x3)`. See `?te` for more information.

4.1.7 Further Interpretable Structure

Viewing model smooth functions of more than one variable as statistical interaction terms leads naturally to structuring models in terms of 'main' smooth effects plus interactions, where the interaction now includes only smooth variation not captured by the main effects. For example we might be interested in smooth model terms of the form

$$f_j(x_j) + f_k(x_k) + f_{jk}(x_j, x_k),$$

where f_{jk} should now be constructed to exclude functions of the form $f_j(x_j) + f_k(x_k)$. Such a construction turns out to be very easy. Let's assume that the bases and penalties for f_j and f_k will be used as the marginal smooths for f_{jk} (although it's not strictly necessary to do so). If we remove the constant function from the basis for f_j, then we exclude the basis for the univariate effect f_k from the basis for f_{jk}, and eliminating the constant function from the f_k basis similarly removes the univariate f_j basis. So all we need do is to apply the sum-to-zero indentifiabilty constraints to the marginal bases (and penalties) *before* using them to construct f_{jk} and the resulting basis has the desired property, with penalties whose

interpretation is unchanged (the resulting basis requires no further identifiability constraint, of course). In `mgcv` terms like `ti(x1,x2)` are used to construct such interactions without main effects (so something like `s(xj)+s(xk)+ti(xj,xk)` would implement the example above). Note that a term like `ti(x1,x2,x3)` excludes both main effects and lower order (i.e. second, here) interactions.

Of course we should not expect a model formulated this way to give identical results to one in which we do not specify main effects and interactions as separate smooths, and use a single tensor smooth constructed from unconstrained marginal bases. The space of functions defined by the model bases is identical in both cases, but the penalty is obviously not. There are 4 smoothing parameter in the main effects + interactions representation, rather than 2, for example.

Another common approach is to consider interactions of smooth and parametric effects. If the parametric term is the linear effect of a metric predictor variable then we get a so called *varying coefficient* term [17], such as $f(x_j)x_k$. Essentially the slope parameter associated with x_k varies smoothly with x_j (often time). When the smooth is a function of spatial location, e.g. $f(\texttt{lon}_i, \texttt{lat}_i)x_i$ then the model is sometimes refered to as 'geographic regression'. The obvious interaction of a smooth effect with a factor variable produces a separate smooth for each factor level—or alternatively one smooth for reference level of the factor, and then a 'difference' smooth for each remaining level.

Any linear functional of a smooth is also readily incorporated into the model. For example suppose that we are interested in the effect of temperature on electricity load over several preceding days. One possibility is to use terms like

$$\sum_{j=1}^{J} f_j(T_{i-j})$$

where each f_j is a separate smooth function and T_i is the mean temperature on day i. But is the effect of yesterday's temperature going to look entirely different to the effect of the day before yesterday's temperature? It seems more likely that the shape of the f_j would change smoothly with j. In that case we could instead use the model term

$$\sum_{j=1}^{J} f(T_{i-j}, j)$$

where f is a tensor product smooth of lagged temperature and the size of the lag. If `T` is an $n \times J$ matrix such that $\texttt{T}_{ij} = T_{i-j}$ and `L` is a matrix of the same dimension such that $\texttt{L}_{ij} = j$ then `te(T,L)` would produce just such a term in `mgcv`. The smooth is simply evaluated at each element of its matrix arguments, to produce a matrix of values, and the columns of this matrix are then summed over, to obtain the contribution to the model's linear predictor.

4.2 From GAM to GAMLSS: Interpretability for Model Building

As explained above, in standard GAM models the conditional distribution $\pi(y|\boldsymbol{x})$ is modelled via a distribution belonging to the exponential family with parameter vector $\boldsymbol{\theta} = \{\mu, \phi\}$. While μ is allowed to vary with the covariates, ϕ does not vary with $\boldsymbol{x}$. This setting can be generalized in two ways. First it is possible to consider distributions that do not belong to the exponential family, and that are parametrised by a general m-dimensional vector $\boldsymbol{\theta}$ controlling (for example) the location, scale and skewness of the distribution. Second, it is possible model each element of $\boldsymbol{\theta}$ via a separate additive model, that is

$$g^l\{\theta_i^l\} = \mathbf{A}_i^l \gamma^l + \sum_j f_j^l(x_{ji}), \quad \text{for} \quad l = 1, \ldots, m. \tag{4.3}$$

Hence, we have l linear predictors, one for each parameter. The simplest example of a such a generalized additive models for location scale and shape (GAMLSS) model [27] is a Gaussian location-scale model where the mean, θ^1, and the variance, θ^2, are both allowed to vary with $\boldsymbol{x}$. In `mgcv` package, such a model can be built and fitted to data via code such as

```
fit <- gam(formula = list(y ~ x1 + s(x2, k = 15, bs = "cr"),
                          ~ s(x2)),
           family = gaulss, data = SomeData)
```

Here the `gaulss` function implements the location-scale version of the Gaussian family. When this family is used, we have to provide `gam` with a list of formulae, the first of which is used to model the mean and the second the variance. Under the `gaulss` family, g_1 is by default the identity function while $g_2(\theta_2) = \log(\theta_2 + b)$, where $\theta_2 = 1/\sigma = \text{var}(y)^{-1/2}$ and b is a small positive constant useful to improve numerical stability. The `mgcv` package provides several other GAMLSS families covering, among others, the Tweedie (`twlss`), generalized extreme value (`gevlss`) and the zero inflated poisson (`ziplss`) distribution.

GAMLSS models are particularly useful when we want to model all the features of $\pi(y|\boldsymbol{x})$, rather than only the conditional mean $\mathbb{E}(y|\boldsymbol{x})$. This might be the case, for example, when forecasting electricity demand in the short term. In fact, the underlying economic loss $L(y, D)$ that characterises a trading or production planning decision D is not necessarily the quadratic loss, which is minimised by $\mathbb{E}(y|\boldsymbol{x})$. Hence, it is preferable to estimate the optimal decision D^*, by modelling the full distribution of $y|\boldsymbol{x}$, using it to estimate $\mathbb{E}\{L(y, D)|\boldsymbol{x}\}$ and minimising the latter w.r.t. D. Adequately modelling $\pi(y|\boldsymbol{x})$ might be important even for applications where $\mathbb{E}(y|\boldsymbol{x})$ is the only quantity of interest. This because the GAM fitting and inferential framework detailed in Sect. 4.1 relies on the model for $\pi(y|\boldsymbol{x})$ being at least approximately correct. Hence, for example, if the variance of y varies wildly with $\boldsymbol{x}$, than the credible intervals obtained by fitting a Gaussian GAM with constant variance might have very poor frequentist coverage.

GAMLSS models provide extra modelling flexibility, but require more effort on the part of the modeller during model selection and checking. In fact, the user has to select which effects to include in each of the m linear predictors. Naïve automatic variable selection is often infeasible, due to the number of effects that could potentially be included and to the fact that GAMLSS models are computationally more expensive to fit than GAMs. If automated model building is key, an attractive option is the `gamboostLSS` `R` package, which implements the gradient boosting GAMLSS fitting methods proposed by Mayr et al. [21].

User-driven, interactive GAMLSS model development is complicated by interpretability issues. For instance, even a very experienced electricity demand modeller might not know which effects should be used to model the skewness of the demand distribution. To mitigate this issue, Fasiolo et al. [11] proposes a visualisation framework, implemented in the `mgcViz` package, which is meant to aid interactive GAMLSS model development and checking. In Sect. 4.2.1, we show how to use `mgcViz` for interactive model building, in the context of electricity demand modelling.

4.2.1 GAMLSS Modelling of UK Aggregate Electricity Demand

4.2.1.1 Data Overview and Pre-processing

This data set contains aggregate UK electricity demand data, at half-hourly resolution and covering the period from the 1st of January 2011 to the 30th of June 2016. The raw demand data was obtained from https://demandforecast.nationalgrid.com. Figure 4.5 shows some of the characteristics of the data. In particular, plot 4.5a shows that the long-term demand trend is negative. This is because the demand contained in the data set is net of embedded production from, e.g., solar panels and wind turbines. That is, net UK demand is decreasing because embedded generation is increasing faster than gross demand. The curves in plot 4.5b show the daily demand profiles for each day of the week. Unsurprisingly, demand is lower on weekends with a delayed morning peak, as people wake up later. The brightness of the hexagonal map in the background is proportional to the number of observations falling within each hexagon. Plot 4.5d shows how demand varies with temperatures, for each day of the week. The effect of temperature is similar on working days, while the heating effect ($t < 10°C$) seems stronger on Saturdays than on Sundays. Figure 4.5c shows that demand is higher during the winter than during the summer (time of year is 0 on Jan 1st and 1 on the 31st of Dec). Between years discrepancies in yearly demand profiles are substantial, especially in the winter.

We integrate the demand data with hourly temperatures, interpolated at the half-hourly resolution, from the NCEI. The temperature data was measured in the proximity of several large UK cities. We built a single temperature variable by

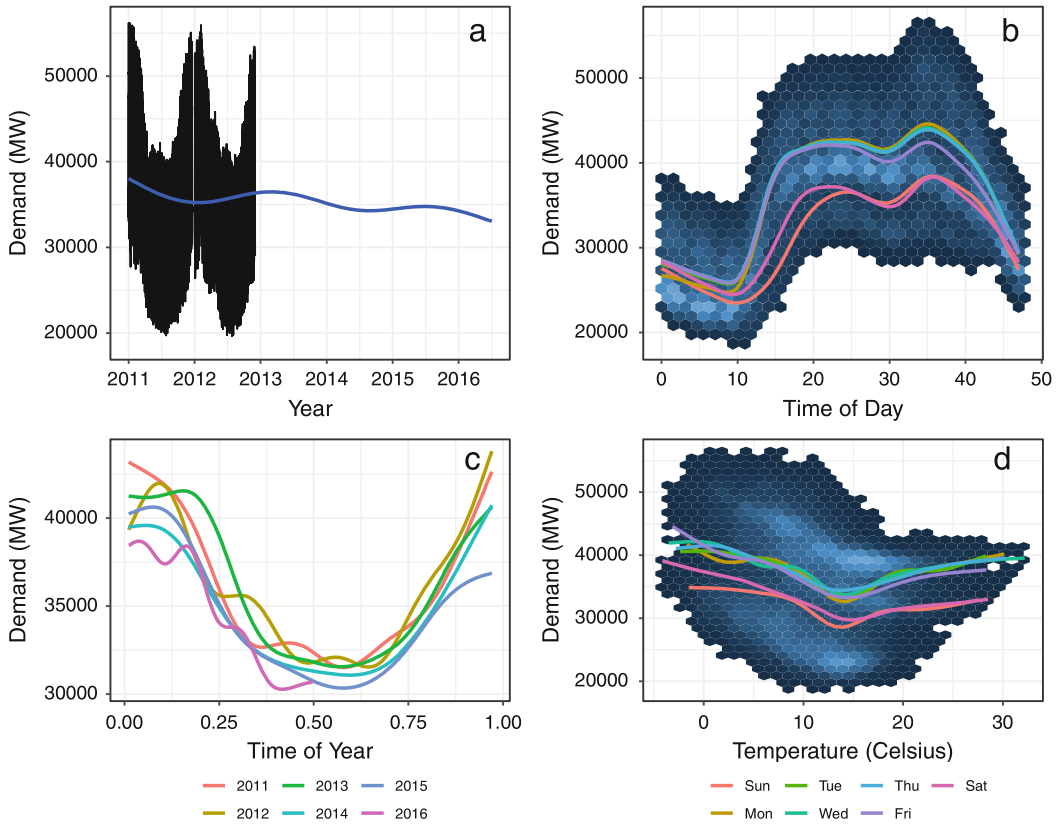


Fig. 4.5 The plots show: (**a**) demand vs time and the long-term trend; (**b**) the daily demand profiles for each day of the week; (**c**) seasonal demand dynamic for each year and (**d**) temperature effect for each day of the week. See the main text for more details

averaging these temperatures with weights that are proportional to the population of each city. Hence, the variable of interest are:

- `timeCount` is a progressive time counter;
- `toy` is the time of year from 0 (1st Jan) to 1 (31st Dec);
- `dow` is a factor variable indicating the day of the week;
- `holy` is a binary variable indicating holidays;
- `tod` is the time of day, ranging from 0 to 47, where 0 indicates the period from 00:00 to 00:30, 1 the period from 00:30 to 01:00 and so on;
- `temp` is the external temperature in degrees Celsius.
- `load48` the demand in the same half-hourly period of the previous day;
- `temp95` an exponential smooth of `temp`, that is `temp95[i] = a*temp[i] + (1-a)*temp95[i-1]` with `a = 0.05`.

The time period covered by the data set contains several exceptional days, on which demand cannot reasonably be modelled on the basis of historical data. In particular, we remove data corresponding to the 29th of April 2011 (Royal Wedding) and to the 4th–5th of June 2012 (Queen Elizabeth's Diamond Jubilee). We exclude also bank holidays and the days in the proximity of Easter, in particular: 22nd to 25 of April 2011, 6th to 9th of April 2012, 29th of March to 1st of April 2013, 18th to 21st of April 2014, 3rd to 6 of April 2015 and 25th to 28th of March 2016. The Christmas and New Year period is similarly difficult to forecast solely on the basis of historical data (that is, without manual or anyway ad hoc intervention), hence we remove data between 21st of December and the 4th of January of each year. Removal of these data leads to the vertical white stripes visible in Fig. 4.5a. We have not excluded data corresponding to the 2012 London Summer Olympic Games, but this is clearly an exceptional period as well.

4.2.1.2 Interactive GAMLSS Model Building

The data is contained in the `electBook` package, available at github.com/mfasiolo/electBook. We start by loading it:

```
library(electBook)
data(UKL)
```

The first model we consider is a simple Gaussian GAM model, with linear effects for the time trend, the lagged load and the day of the week. We include smooth effects for (smoothed) temperature, the time of day and the time of year. For the last two we use cyclical smooths (`bs = "cc"`). We use the `knot` argument to ensure that, for `toy`, the effect on the 1st of January matches the effect on the 31st of Dec while for `tod` the effect at 00:00 to 00:30 matches the same effect at the same time of the following day.

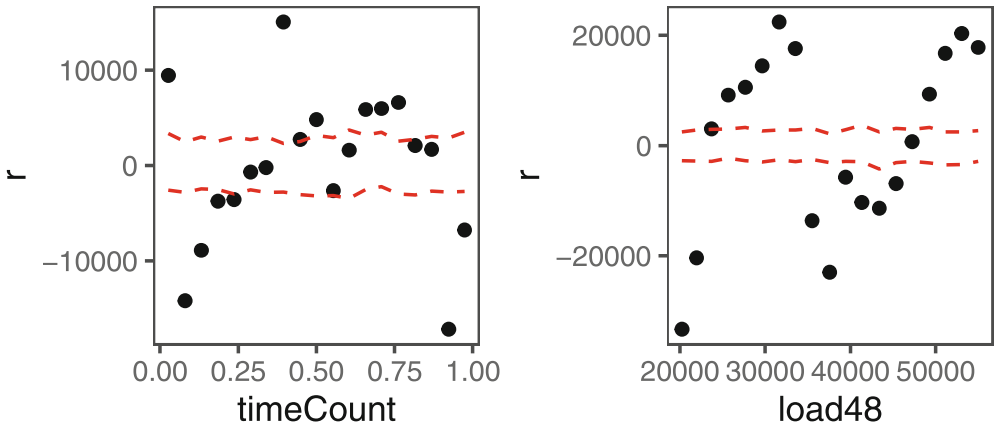
```
library(mgcv)
fit0 <- gam(load ~ timeCount + load48 + dow + s(temp) + s(temp95) +
                   s(tod, bs = "cc") + s(toy, bs = "cc"),
            knots = list(tod = c(0, 48), toy = c(0, 1)), data = UKL)
```

To use the methods provided by `mgcViz` to visualise the fitted GAM, we need to load the package and to convert the output of `gam` to an object of class `gamViz`, via the `getViz` function:

```
library(mgcViz)
fit0 <- getViz(fit0, nsim = 100)
```

The arguments `nsim` is used to specify that we want to simulate 100 residuals vectors from the fitted GAM, which we use within the diagnostics offered by `mgcViz`. In particular, here we use `check1D` and `l_gridCheck1D` to check the residuals along the `timeCount` and `load48` covariates:

```
pl0 <- check1D(fit0, list("timeCount", "load48", "toy")) +
       l_gridCheck1D(level = 0.95, showReps = FALSE)
print(pl0, pages = 1, layout_matrix = matrix(1:3, 1, 3))
```



The black points are the mean of the observed residuals from our model, binned depending on the value of the variable of the x axis. The red lines are 95% reference intervals, estimated by binning and averaging the simulated residuals, and computing the empirical quantiles of the binned means. The plots show that the residuals deviate significantly from what we would expect under the model. In particular, the plots suggest that the effects of `timeCount` and `load48` are non-linear, while the fact that the mean residuals are negative for `toy` ≈ 0 and positive for `toy` ≈ 1 implies that using a cyclical smooth for `toy` is not appropriate.

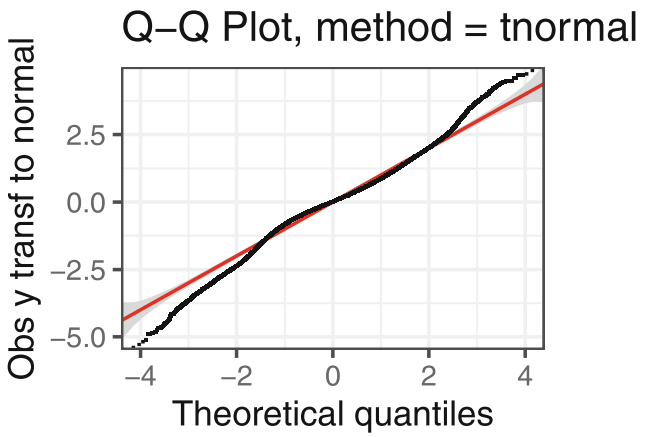
We can check whether `k` is large enough via:

```
check(fit0)

  ...
  ##             k'  edf k-index p-value
  ## s(temp)   9.00 8.03    0.98    0.09 .
  ## s(temp95) 9.00 7.87    0.97    0.03 *
  ## s(tod)    8.00 8.00    0.96  <2e-16 ***
  ## s(toy)    8.00 7.72    0.94  <2e-16 ***
  ## ---
  ## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
  ...
```

Similarly to `gam.check`, the code above produces some visual residual checks (not shown) and the table above. The table (see Sect. 4.1.5 for explanations on how to interpret it) suggests that we should consider using more basis functions for `tod` and `toy`. As a further check we can look at a QQ-plot:

```
qq(fit0, method = "tnormal", CI = "normal", level = 0.95)
```



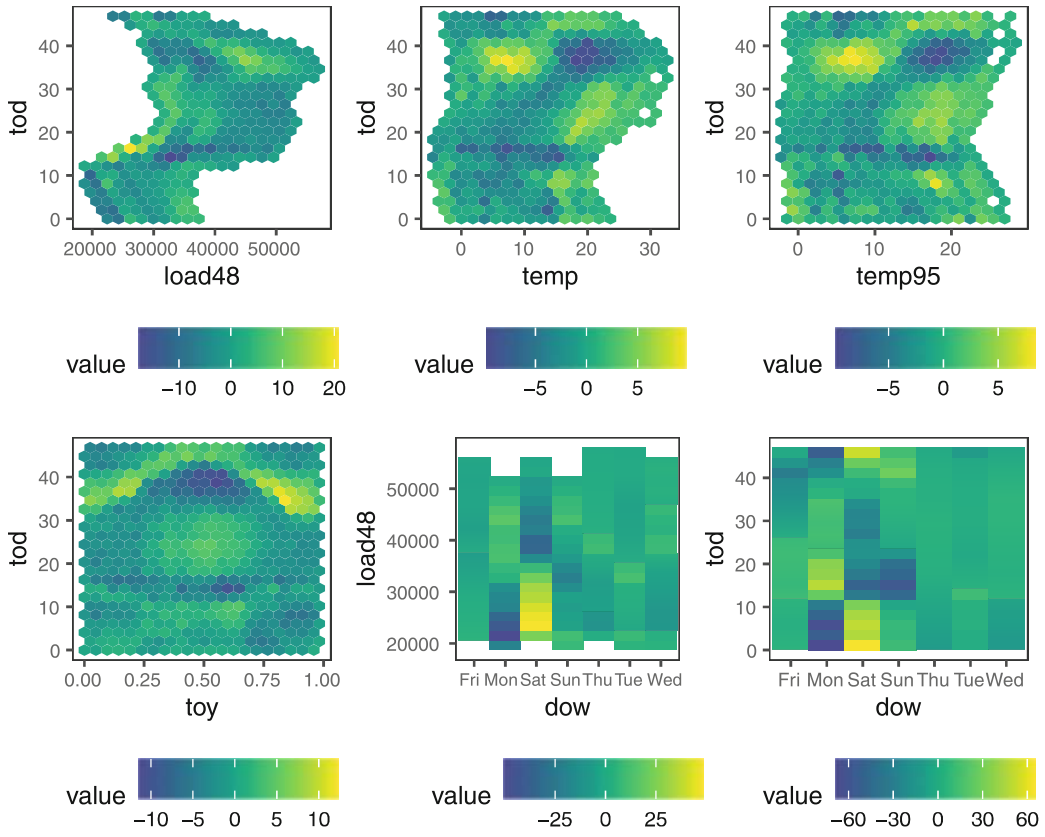
which shows that the residuals distribution has heavier tails than the Gaussian model used here. Hence, here we fit a more robust GAM based on a scaled Student's t distribution:

```
fit1 <- gam(load ~ dow + s(timeCount, k = 4) + s(load48) + s(temp) +
                   s(temp95) + s(tod, k = 20, bs = "cc") + s(toy, k = 20),
                 data = UKL, family = scat,
                 knots = list(tod = c(0, 48)))
fit1 <- getViz(fit1, nsim = 100)
```

which also implement the other improvements we identified above, that is (a) smooth effects for `timeCount` and `load48`, (b) a larger basis dimension for `tod` and `toy`, (c) remove the cyclical constraint from `s(toy)`.

With almost 10^5 observations, the `UKL` is large enough to consider including smooth interactions. The `mgcViz` package provides functions specifically designed to aid visual interaction detection. In particular, consider the following code:

```
pl <- check2D(fit1, x1 = list("load48", "temp", "temp95", "toy", "dow", "dow"),
                    x2 = list("tod", "tod", "tod", "tod", "load48", "tod")) +
      l_gridCheck2D( )
print(pl + theme(legend.position="bottom"), pages = 1)
```



The heat maps are obtained by averaging the residuals falling within each hexagonal bin, and standardising the binned means via the binned simulated residuals. If the binned means are roughly normally distributed and the model is not missing any interaction, then we should expect most standardised means to fall in $(-2, 2)$, which is approximately a 95% Gaussian reference interval. But the colour scales show that the observed deviations are much larger than that. Further, the heat maps show some non-random patterns, which could be modelled via smooth interactions. The plots of `load48` and `tod` vs `dow` show that these effects depend on the day of the week. In particular, the plot on the bottom right suggests that the daily demand profile is different on Monday, Saturday and Sunday, relative to each other and to the rest of the week. To capture this in an improved model, we recode the `dow` variable as follows:

```
library(dplyr)
UKL$dowGroup <- recode_factor(UKL$dow, Tue = "Tue-Fri", Wed = "Tue-Fri",
                                       Thu = "Tue-Fri", Fri = "Tue-Fri")
levels(UKL$dowGroup)

   ## [1] "Tue-Fri" "Sun"     "Mon"     "Sat"
```

Checking again whether the bases dimensions are large enough:

```
check(fit1)

   ...
   ##                  k'   edf k-index p-value
   ## s(timeCount)   3.00  2.95    0.83  <2e-16 ***
   ## s(load48)      9.00  8.96    0.97   0.010 **
   ## s(temp)        9.00  7.48    1.00   0.605
   ## s(temp95)      9.00  7.45    0.99   0.310
   ## s(tod)        18.00 17.89    0.97   0.045 *
   ## s(toy)        19.00 17.57    0.93  <2e-16 ***
   ## ---
   ## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
   ...
```

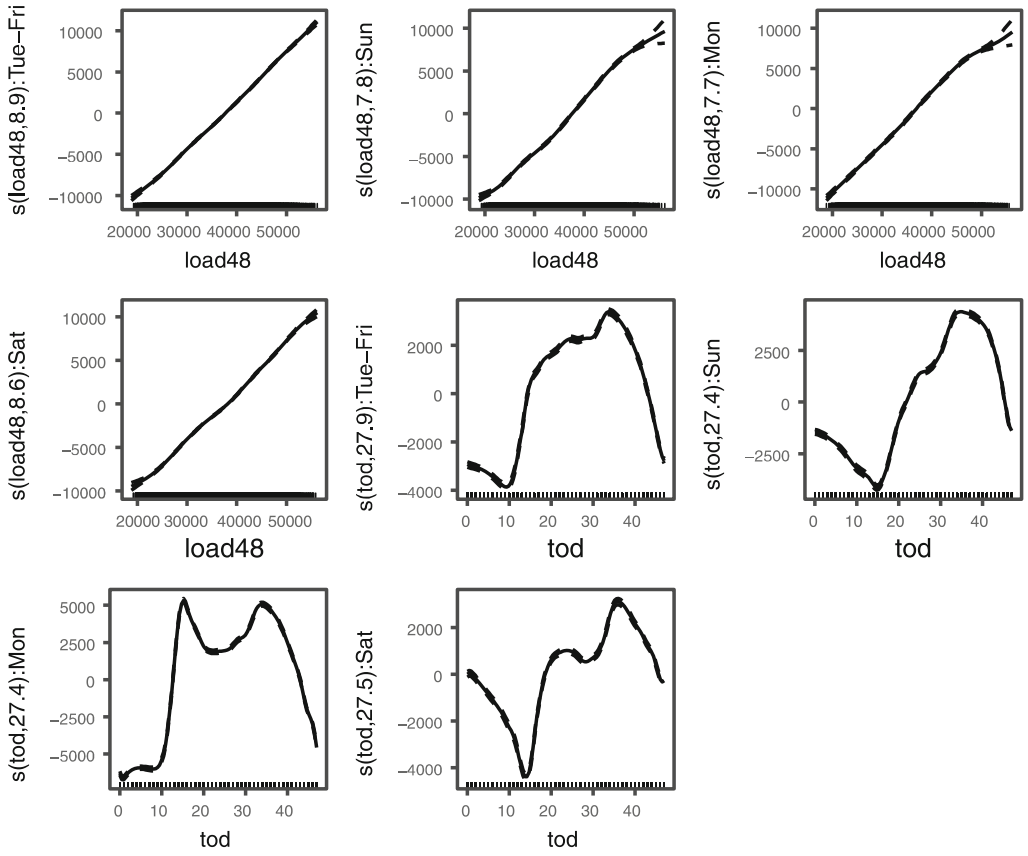
suggests that we should increase the value of `k` used for the effects of `tod`, `toy`, `timeCount` and `load48`. Increasing `k` for the first two effects is reasonable, but doing so for `timeCount` is risky, because the residuals of our model are auto-correlated. Residual auto-correlation is not modelled here, hence increasing `k` for the effect of `timeCount` would lead to a very wiggly effect, which extrapolates badly into the future. Hence, for this effect the value of `k` should not be chosen on the basis of the table above, but on other considerations. In particular, here the effect of `timeCount` is meant to capture long-term changes in `load`, due to technological and social changes that take place over the years. Hence, a small value of `k` is appropriate. We could consider increasing the value of `k` for `load48` but, from an interpretability point of view, it is preferable to have an auto-regressive effect that is close to linear.

On the basis of the considerations above, we fit the following model:

```
fit2 <- bam(load ~ dow + s(timeCount, k = 4) +
                     s(load48, k = 10, by = dowGroup, id = 1) +
                     s(tod, k = 30, bs = "cc", by = dowGroup, id = 2) +
                     s(temp, k = 20) + s(temp95, k = 20) +
                     s(toy, k = 30) +
                     ti(load48, tod, k = c(5, 5), bs = c("cr", "cc")) +
                     ti(temp, tod, k = c(5, 5), bs = c("cr", "cc")) +
                     ti(temp95, tod, k = c(5, 5), bs = c("cr", "cc")) +
                     ti(toy, tod, bs = c("cr", "cc"), k = c(5, 5)),
                   data = UKL,
                   family = scat,
                   knots = list(tod = c(0, 48)),
                   discrete = TRUE)
fit2 <- getViz(fit2, nsim = 100)
```

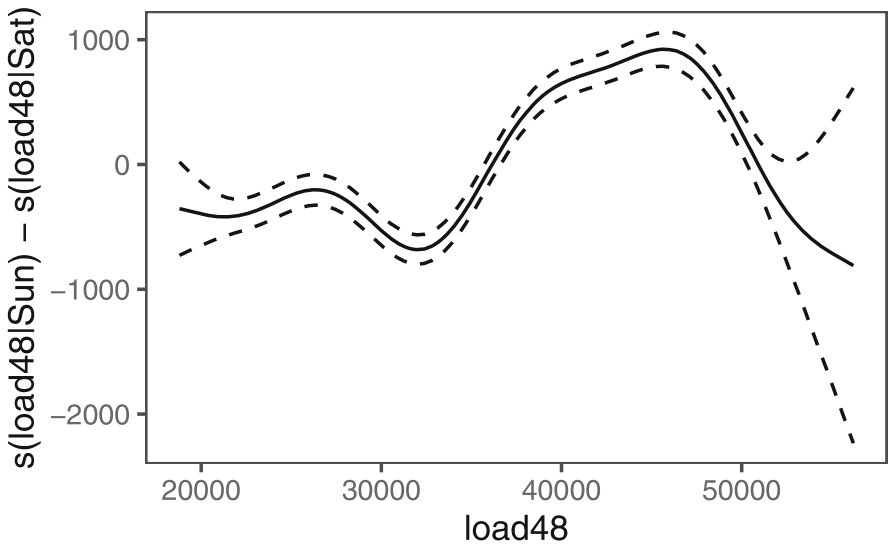
The model is fitted via the `bam` function, which implements the big data GAM methods of [35]. In particular, setting `discrete = TRUE` leads to faster computation via covariate discretisation. In `s(load48, k = 10, by = dowGroup, id = 1)` the `by` argument allows us to create smooth effects of `load48` that are different depending on the value of the `dowGroup` factor. By setting `id = 1` we impose that wiggliness of the by-factor smooths should be penalised via a single smoothing parameter, rather than one for each factor level.

Let us look at all the by-`dowGroup` effects:



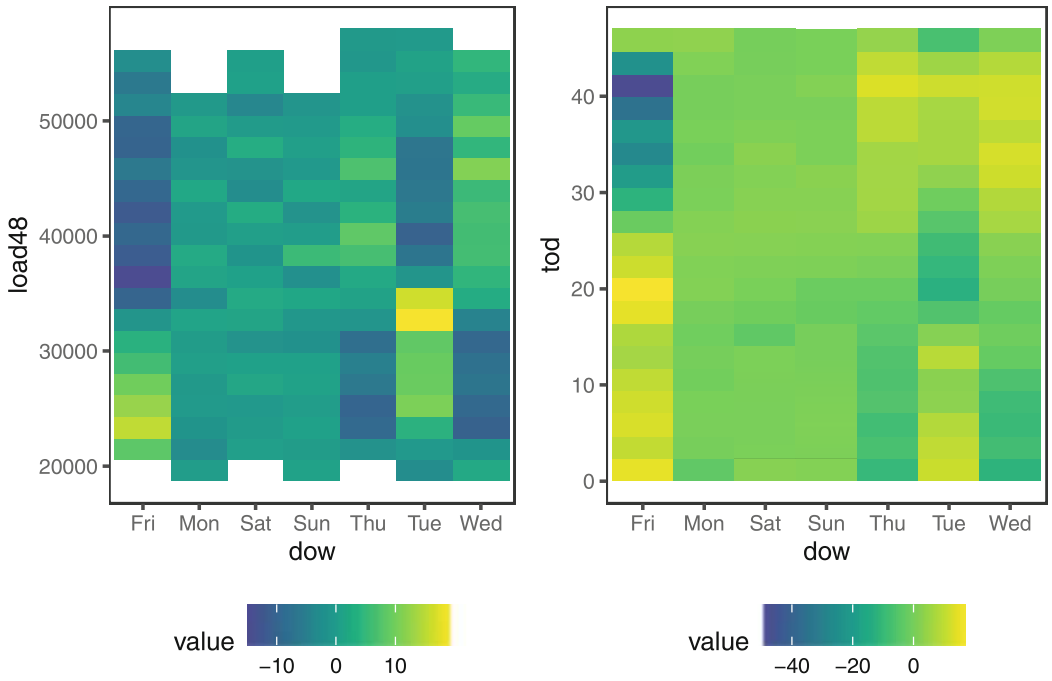
The effect of `tod` is clearly highly dependent on `dowGroup`. This is not surprising, for instance most people wake up later on Sunday than on working says, hence the morning demand ramp is gentler on Sundays. It is hard to visually assess whether the effect of `load48` is different depending on `dowGroup`. The difference can be visualised via the following plot:

```
plotDiff(sm(fit2, 3), sm(fit2, 5)) + ylab("s(load48|Sun) - s(load48|Sat)")
```



Under the `dowGroup` grouping, the weekdays between Tuesday and Friday are grouped together hence our model assumes that the effects of `load48` and `tod` do not vary strongly between those days. We can check whether this is the case via a further interaction check:

```
pl <- check2D(fit2, list("dow", "dow"), list("load48", "tod")) + l_gridCheck2D()
print(pl + theme(legend.position="bottom"), pages = 1)
```

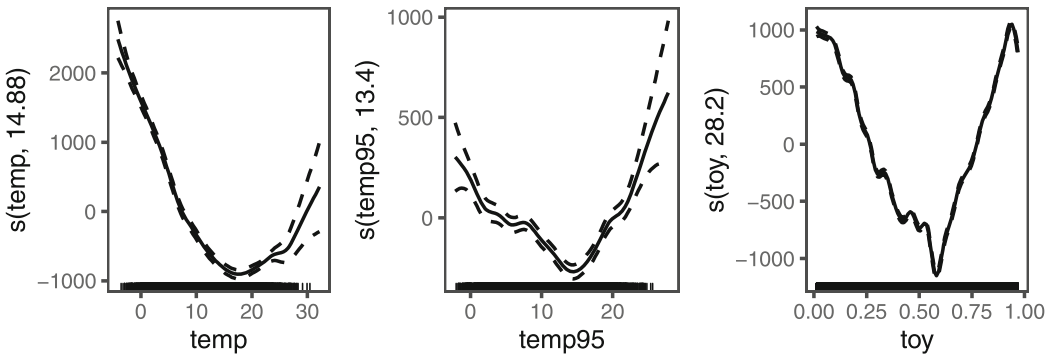


The left plot suggests that the effect of `load48` is significantly different depending on the day of the week. Strong residuals pattern are visible on Friday and Tuesday. The right plot show that we are grossly over-estimating the demand on Friday night. This suggest that grouping together the days between Tuesday and Friday might not

have been a great idea, and that we might be better of specifying by-factor effects that are different for each day of the week (`dow`).

Let us looks at the temperature and time of year effects:

```
print(plot(fit2, select = 10:12), pages = 1, layout_matrix = matrix(1:3, 1))
```



The plot for `toy` shows a sharp trough in the summer (`toy`$\approx$ 0.6) and a narrow peak in late winter. In particular, the effect seems to be more wiggly in some periods than in others. We could accommodate for this by using an adaptive smooth (`bs = "ad"`), under which the smoothness penalty varies with the variable of interest (`toy` here).

The following model:

```
fit3 <- bam(load ~ dow + s(timeCount, k = 4) +
                     s(load48, k = 10, by = dow, id = 1) +
                     s(tod, k = 30, bs = "cc", by = dow, id = 2) +
                     s(temp, k = 20) + s(temp95, k = 20) +
                     s(toy, bs = "ad", k = 30) +
                     ti(load48, tod, k = c(5, 5), bs = c("cr", "cc")) +
                     ti(temp, tod, k = c(5, 5), bs = c("cr", "cc")) +
                     ti(temp95, tod, k = c(5, 5), bs = c("cr", "cc")) +
                     ti(toy, tod, bs = c("cr", "cc"), k = c(5, 5)),
                   data = UKL,
                   family = scat,
                   knots = list(tod = c(0, 48)),
                   discrete = TRUE)
fit3 <- getViz(fit3, nsim = 100)
```

integrates the improvement identified above, namely an adaptive smooth for `toy`, and by-`dow` smooths for `load48` and `tod`. The model achieves the lowest AIC so far:

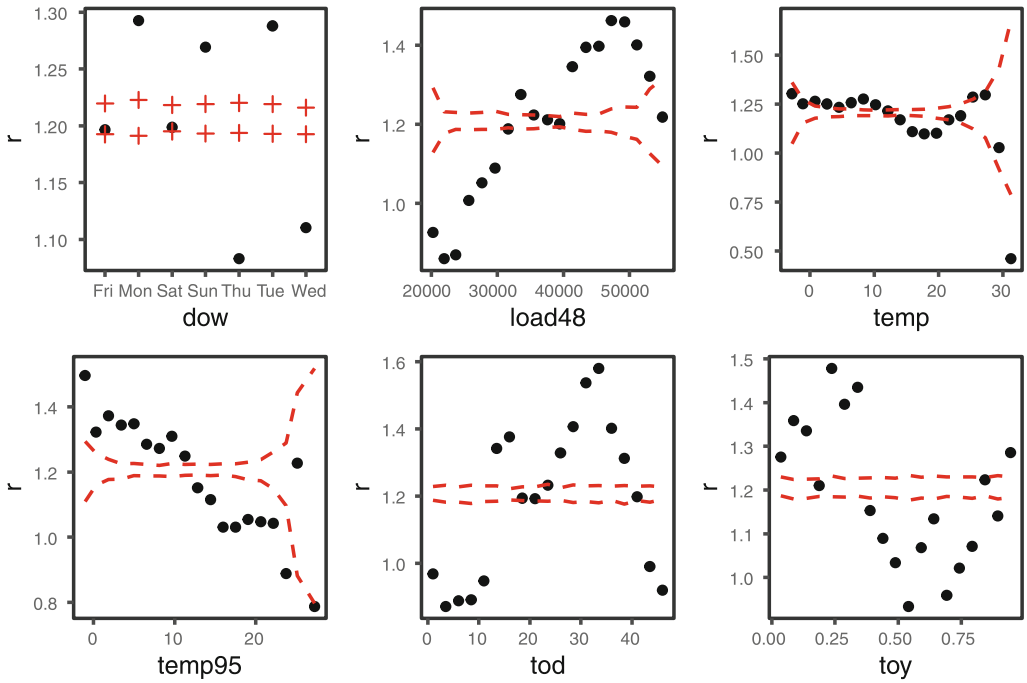
```
AIC(fit0, fit1, fit2, fit3)
```

```
##              df     AIC
## fit0  41.62090 1584219
## fit1  73.20387 1569250
## fit2 261.01765 1432123
## fit3 370.42144 1418360
```

One could consider improving it further, for example by checking whether the number of basis function used to buing the tensor product interactions is sufficiently large. However, below we assume that the last GAM above models the conditional mean demand sufficiently well and we move now to the task of modelling the whole conditional demand distribution in a GAMLSS framework.

We start by looking at how the standard deviation of the residuals varies with some of the covariates:

```
pl0 <- check1D(fit3, list("dow", "load48", "temp", "temp95", "tod", "toy")) +
  l_gridCheck1D(gridFun = sd, level = 0.95, showReps = FALSE)
print(pl0, pages = 1)
```



The plots are obtained by binning observed and simulated residuals as before, but now we are summarising the binned residuals via `sd`, rather than `mean`. The plots show that the residual variance varies wildly with most covariates. Importantly, the variance changes smoothly with the continuous covariates, which suggests that we could model these patterns via smooth effects.

To model the variance as well as the mean, we consider the following Gaussian location-scale model:

```
fit_lss <- gam(list(load ~ dow + s(timeCount, k = 4) +
                      s(load48, k = 10, by = dow, id = 1) +
                      s(tod, k = 30, bs = "cc", by = dow, id = 2) +
                      s(temp, k = 20) + s(temp95, k = 20) +
                      s(toy, bs = "ad", k = 30) +
                      ti(load48, tod, k = c(5, 5), bs = c("cr", "cc")) +
                      ti(temp, tod, k = c(5, 5), bs = c("cr", "cc")) +
                      ti(temp95, tod, k = c(5, 5), bs = c("cr", "cc")) +
                      ti(toy, tod, bs = c("cr", "cc"), k = c(5, 5)),
                    ~ dow + s(load48) + s(temp95) + s(tod, k = 20) + s(toy)),
               data = UKL,
               family = gaulss,
               knots = list(tod = c(0, 48)))
fit_lss <- getViz(fit_lss, nsim = 100)
```

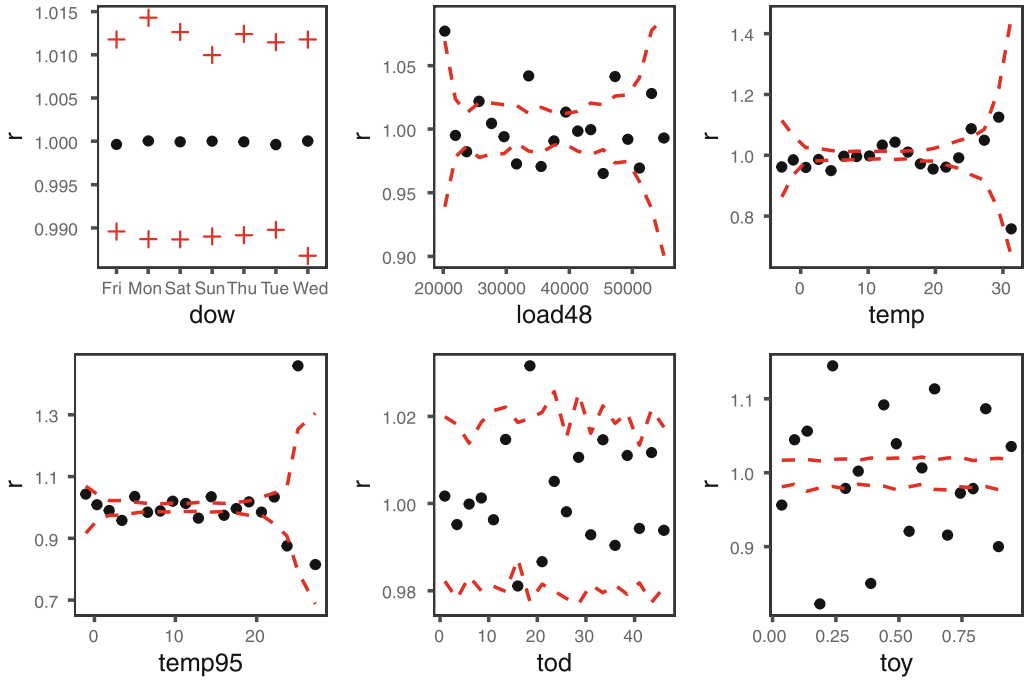
The new model is preferable in terms of AIC:

```
AIC(fit3, fit_lss)

  ##                df      AIC
  ## fit3    370.4214 1418360
  ## fit_lss 419.7592 1410087
```

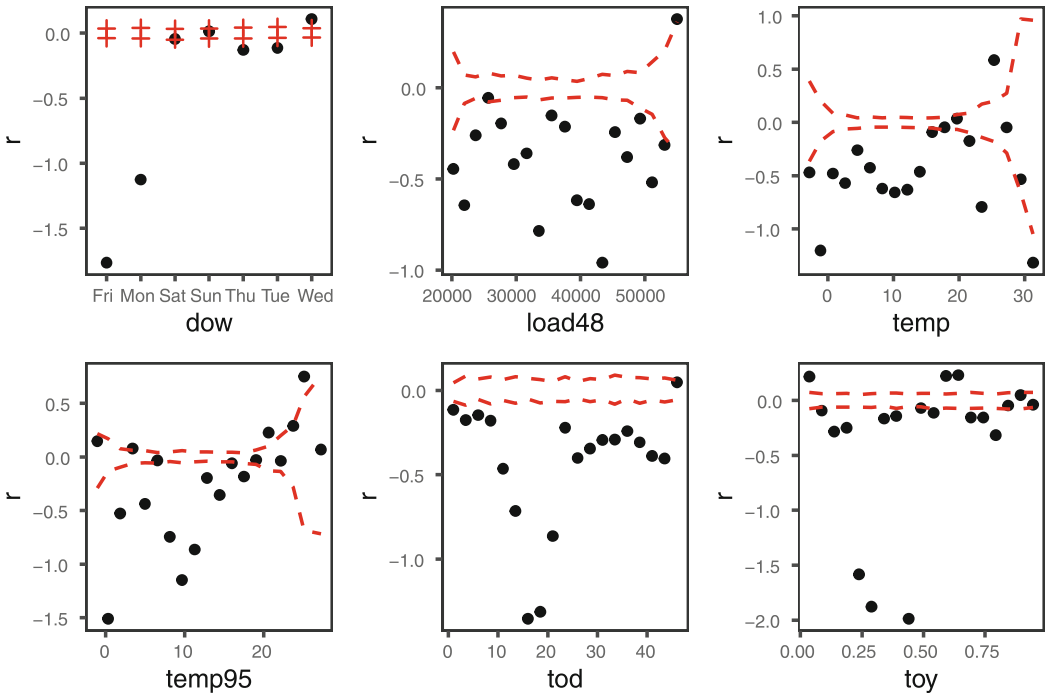
and the residual variance patterns are now much weaker:

```
pl0 <- check1D(fit_lss, list("dow", "load48", "temp", "temp95", "tod", "toy")) +
  l_gridCheck1D(gridFun = sd, level = 0.95, showReps = FALSE)
print(pl0, pages = 1)
```



As we did for the mean model earlier, we could now verify whether the number of basis functions used to model the effects in the variance model is sufficiently large and we could look for bivariate interactions. While we leave this to the interested reader, here we verify whether further features of the conditional response distribution are correctly captured by the model. In particular, we look at how residual skewness varies with the covariates:

```
library(e1071) # Provides the skewness() function
pl0 <- check1D(fit_lss, list("dow", "load48", "temp", "temp95", "tod", "toy")) +
  l_gridCheck1D(gridFun = skewness, level = 0.95, showReps = FALSE)
print(pl0, pages = 1)
```



Interestingly, the plots show that, on average, the residuals are skewed to the left. We see this from the fact that most observed binned skewness values fall below zero. Of course, skewness can not be model via a Gaussian distribution, which is symmetric. Further, the skewness varies with most covariates. For instance, the residual distribution is left skewed on Mondays and Fridays, and in the mornings (`tod`≈20). The skewness patterns are less smooth than the variance patterns observed above, hence we can not expect smooth effects to capture them as well as they did for the variance.

To model the demand skewness, we consider the four-parameter sinh-arcsinh distribution of [19]. The model allows us to control the location, scale, asymmetric and tail behaviour of the distribution via separate parameters. While we could have a full additive model for each, here we consider the following model:

```
fit_shash <- gam(list(load ~ dow + s(timeCount, k = 4) +
                        s(load48, k = 10, by = dow, id = 1) +
                        s(tod, k = 30, bs = "cc", by = dow, id = 2) +
                        s(temp, k = 20) + s(temp95, k = 20) +
                        s(toy, bs = "ad", k = 30) +
                        ti(load48, tod, k = c(5, 5), bs = c("cr", "cc")) +
                        ti(temp, tod, k = c(5, 5), bs = c("cr", "cc")) +
                        ti(temp95, tod, k = c(5, 5), bs = c("cr", "cc")) +
                        ti(toy, tod, bs = c("cr", "cc"), k = c(5, 5)),
                      ~ dow + s(load48) + s(temp95) + s(tod, k = 20) + s(toy),
                      ~ dow + s(load48) + s(temp95) + s(tod) + s(toy),
                      ~ 1),
                 data = UKL,
                 family = shash,
                 knots = list(tod = c(0, 48)))
fit_shash <- getViz(fit_shash, nsim = 100)
```

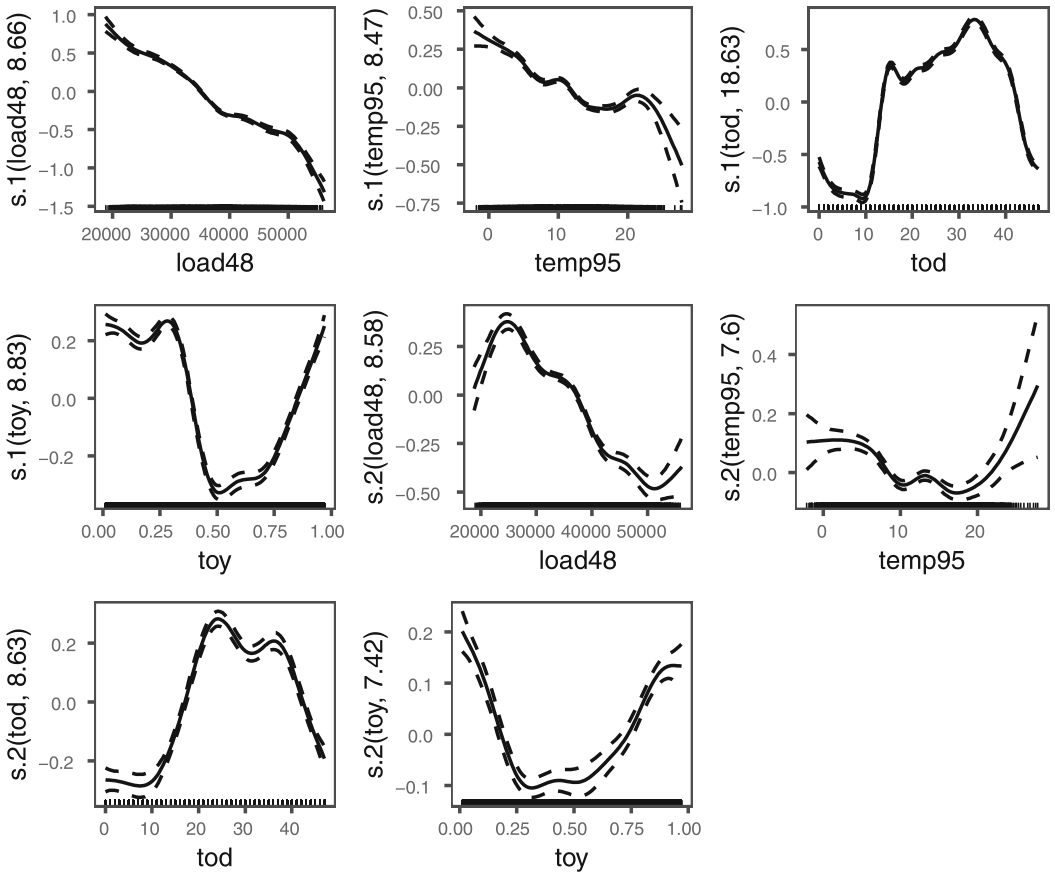
which assumes that the fourth parameter, which controls the tail behaviour, does not depend on the covariates. One could consider checking whether the kurtosis of the residuals varies with the residuals, but we leave this to the interested reader. This final model is the best in terms of AIC:

```
AIC(fit_lss, fit_shash)

  ##                  df     AIC
  ## fit_lss    419.7592 1410087
  ## fit_shash 462.7579 1403485
```

It is interesting to visualise the fitted effects in the scale and skewness models:

```
print(plot(fit_shash, select = 23:30), pages = 1)
```



The effects on the variance are labelled `s.1` and those on the skeweness `s.2`. Some effects are quite interpretable. For example, the plots for `s.1(tod)` and `s.1(toy)` show that the variance is minimal during the night and in the summer, which is reasonable. The plots on for `s.2(tod)` and `s.2(toy)` show that the demand is skewed to the right in the middle of the day and in the winter, which is also expected. It is more difficult to interpret the effects of lagged demand and of smoothed temperature of conditional variance and skewness.

In this section we considered an electricity demand forecasting application and we explained how to construct increasingly complex GAMLSS models in an interpretable manner, by exploiting the visualisation tools provided by `mgcViz`. However, electricity demand dynamics are complex and ever-changing, due to socio-economical factors and technological change, hence even a carefully crafted GAMLSS model might not be able to model electricity demand in all circumstances. The issue is considered in the next section, where we explain how additive and other types of models can be combined to produce an improved forecasting model, while retaining a degree of interpretability.

4.3 From GAMs to Aggregations of Experts, Are We Still Interpretable?

One important concern in time series forecasting and in particular in electricity load forecasting is due to changes in $\pi(y|\boldsymbol{x})$ with time, that is the data are not i.i.d.. Examples of such changes can be found recently due to the COVID 19 pandemic where a lot of human activites were considerably affected: mobility [24, 25], economic activities [22], pollution [20] and energy consumption [18]. It entailed new challenges in the electricity demand forecasting field. A recent forecasting competition [10] has been set up where the participants faced with the problem of forecasting electricity load consumption during the COVID lock-down in a big city (unknown location). Two of the three top teams [8, 36] applied online aggregation of experts approaches. Online robust aggregation of experts [6, 7] is a powerful model free approach aiming at learning in a streaming fashion characteristics of time series data. It consists in combining forecasts (called experts) according to their past performances. The methods are designed to be efficient even in an adversarial setting, that is where the data generating process can come from an adversary, making them very robust. In a changing context like the one induced by COVID 19 pandemic, online aggregation allows to adapt to changes in distribution and to track the performance of the best experts.

To produce good aggregated forecasts, it is important to aggregate "diverse" experts, coming for various methods and/or using different sources of information [13]. More generally, in the ensemble methods frameworks (bagging, random forest, boosting, etc.), it is well known that increasing the diversity of the base learners improves the ensemble performance and this can be done implicitly by choosing complementary models, learning models on different subsets/transformation of the data (bootstrapping, changing the temporal or spatial resolution, etc.) or optimizing different loss functions. It can also be done explicitly by optimizing some diversity criteria to force base learners to be diverse [2, 26].

We will focus here on experts generated with two methods that are quite complementary: GAMs and Random Forests (RFs). Whereas RFs are algorithmic based, GAMs are more model oriented, as explained above. GAMs performs well on electricity load data and have the nice property to be understandable by humans, the latter property being of crucial interest for the energy production and distribution planning. The price to pay for this interpretability is that designing a good GAM model requires some statistical background as well as domain specific knowledge. In contrast, RFs are powerful black box methods for modelling complex regression relationships [3] with very little prior knowledge of the problem. A RF regression aims at fitting a generic relationship $y_t = f(\mathbf{x}_t) +$ 'noise' by using an efficient and easy to tune data driven optimization algorithm. RFs are obtained by aggregating an ensemble of base learners generated by applying classification and regression trees [4] on different subsets of the data, obtained with bagging and random sampling of covariates. RFs capture well complex non-linear interactions, can be easily adapted in a time varying environment but their forecasts are by nature restricted to the

convex envelop of the training data. However, GAMs allow to explicitly control the smoothness of the functional relationships and have good extrapolation properties. A nice compromise between the two approaches can be obtained by stacking them, as done in [14].

To stack a GAM and a RF we proceed in three steps:

1. Fit a GAM model to the training set and extract the estimated effects $\widehat{f}_j(x_j)$. We denote $\widehat{\mathbf{f}}(\mathbf{x})$ the set of these effects.
2. Estimate the forecasting residuals (either by cross-validation, block cross-validation or forecasting errors in an online forecasting setting) denoted $\widehat{\varepsilon}_t$.
3. Fit a RF model to predict the GAM residuals: $\widehat{\varepsilon}_t = g(\mathbf{x}_t, \widehat{\mathbf{f}}(\mathbf{x})) + u_t$. Hence, the set of features used by this RF includes the original features and the evaluated GAM effects. The final forecasts are obtained summing the GAM forecasts and the corrections provided by the RF.

To fit the RFs, we use the `ranger()` function from the `ranger` `R` package. The default parameters are used (in particular, 500 trees, $mtry = \sqrt{p}$, unlimited tree depth). For the GAMs we use `mgcv`. We also try a variant of RF presented in [15], using moving block bootstrap as an alternative to the classical bootstrap step of the RF algorithm. This is implemented in the `rangerts` package available at https://github.com/hyanworkspace/rangerts.

4.3.1 Online Forecasting with Online Aggregation of Experts

Here we present the experts that we will use in the aggregation and how to compute them on the UK dataset described in Sect. 4.2.1.1. We choose to focus on a subset of this dataset corresponding to `tod = 40`, i.e. 8 pm. We do that for two reasons: to reduce computational time but also because fitting the models and the aggregation to each half-hour of the day usually improve the forecasting performance. We split the data in two, the observations before the 1st of September 2015 are used to train the experts, while the following 281 days are used to the evaluate the online aggregation:

```
library(electBook)
data(UKL)
selH <- which(UKL$tod==40)
UKL_h <- UKL[selH, ]

n0 <- which(as.character(UKL_h$date)=="2015-09-01 20:00:00")
UKL_h0 <- UKL_h[1:n0, ]
UKL_h1 <- UKL_h[-c(1:n0), ]
```

The first expert, called `g0`, is inspired by the simplest GAM model from Sect. 4.2.1.2:

```
g0 <- gam(load ~ s(timeCount, k=4) + load48 + dow + s(temp) + s(temp95) +
                 s(tod, bs = "cc") + s(toy, bs = "cc"),
          knots = list(tod = c(0, 48), toy = c(0, 1)), data = UKL)
g0.forecast  <-  predict(g0, newdata=UKL_h1)
```

The second one, called `g1`, is obtained using a similar equation, but fitted on the single instant 40:

```
g1 <- gam(load ~ s(timeCount, k=4) + load48 + dow + s(temp) + s(temp95) +
                 s(toy, bs = "cc"),
          knots = list( toy = c(0, 1)), data = UKL_h0)
g1.forecast  <-  predict(g1, newdata=UKL_h1)
```

The third and fourth experts are RFs fitted to data from 8 pm only, with the same covariates the GAMs. `rf0` is a basic RF and `rf1` is a moving block bootstrap variant with a block size of 3 weeks:

```
library(ranger)
library(rangerts)
set.seed(350)
rf0 <- ranger(load ~ timeCount + load48 + dow + temp + temp95 + toy,
              data = UKL_h0)
rf0.forecast  <-  predict(rf0, data=UKL_h1)$prediction

rf1 <- rangerts(load ~ timeCount + load48 + dow + temp + temp95 + toy,
              data = UKL_h0, bootstrap.ts= "moving", block.size=7*3)
rf1.forecast  <-  predict(rf1, data=UKL_h1)$prediction
```

The fifth expert is the RF stacking model, using the GAM effects of `g1` as supplementary covariates. We call it `rfgam`:

```
terms0 <- predict(g1, newdata=UKL_h0, type='terms')
terms1 <- predict(g1, newdata=UKL_h1, type='terms')
colnames(terms0) <- paste0("gterms_", c(1:ncol(terms0)))
colnames(terms1) <- paste0("gterms_", c(1:ncol(terms1)))

Data_rf0 <- data.frame(UKL_h0, terms0)
Data_rf0$res <- UKL_h0$load-g1$fitted.values

Data_rf1 <- data.frame(UKL_h1, terms1)
Data_rf1$res <- UKL_h1$load-g1.forecast

cov <- "timeCount + load48 + dow + temp + temp95 + tod + toy +"
gterm <-paste0("gterms_", c(1:ncol(terms0)))
gterm <- paste0(gterm, collapse='+')
cov <- paste0(cov, gterm, collapse = '+')
formule_rf <- paste0("res", "~", cov)
rfgam<- ranger(formule_rf, data = Data_rf0, importance =  'permutation')
rfgam.forecast  <-  predict(rfgam, data=Data_rf1)$prediction + g1.forecast
```

Table 4.1 Performance of the experts and of the online aggregation

	g0	g1	rf0	rf1	rfgam	best convex	BOA
MAPE	2.09	1.14	1.58	1.59	1.06	1.05	1.03
RMSE	1012.70	590.57	818.28	823.62	557.94	555.06	546.28

We can then compute an online aggregation of experts with the subgradient version of Bernstein Online Aggregation (BOA) algorithm [28] as well as the oracle corresponding to the a posteriori best convex aggregation.

```
library(opera)
experts <-  cbind(g0.forecast, g1.forecast, rf0.forecast, rf1.forecast,
                  rfgam.forecast)
colnames(experts) <- c("g0", "g1", "rf0", "rf1", "rfgam")
agg <- mixture(Y=UKL_h1$load, experts=experts, model='BOA',
               loss.gradient=T)
```

Table 4.1 presents the forecasting performances of the experts and of BOA on the test set, showing the improvement induced by fitting the GAM to data from 8 pm and the further performance improvement obtained via stacking RF and GAM.

As expected, the aggregation performs well, if we consider the performance of the best expert and the best convex aggregation. Figure 4.6 is generated by (we recall that the different experts are: `g0` (simplest GAM), `g1` (GAM by instant), `rf0` (basic random forest), `rf1` (block bootstrapped random forest), `rfgam` (stacked GAM and random forest)):

```
plot(agg, type='plot_weight', dynamic = F)
```

and it represents the evolution of the weights on the test set, and clearly shows the substantial weight given to the stacking forest as well as the dynamic evolution of the weights as functions of time.

4.3.2 Visualizing the Black Boxes

The problem of explaining black box machine learning models has received much attention in past few years [23]. Here we focus on accumulated local effects (ALEs) [1], which are popular methods to visualize black boxes by framing them as additive models. ALE methods is implemented in the `ALEplot R` package which can be used to visualise online aggregations, if a corresponding `predict` function is provided.

Figure 4.7 represents the ALEs associated with the covariates `timeCount`, `load48`, `temp` and `toy` (note that the `timeCount` effect overlaps for `g0` and `g1`, as well as for `rf0` and `rf1`). Notice that ALEs allow us to interpret clearly the effects of covariates on the different models as well as on the aggregation. Regarding the `timeCount` effect, this is constant for `rf0` and `rf1`, which is due to the fact

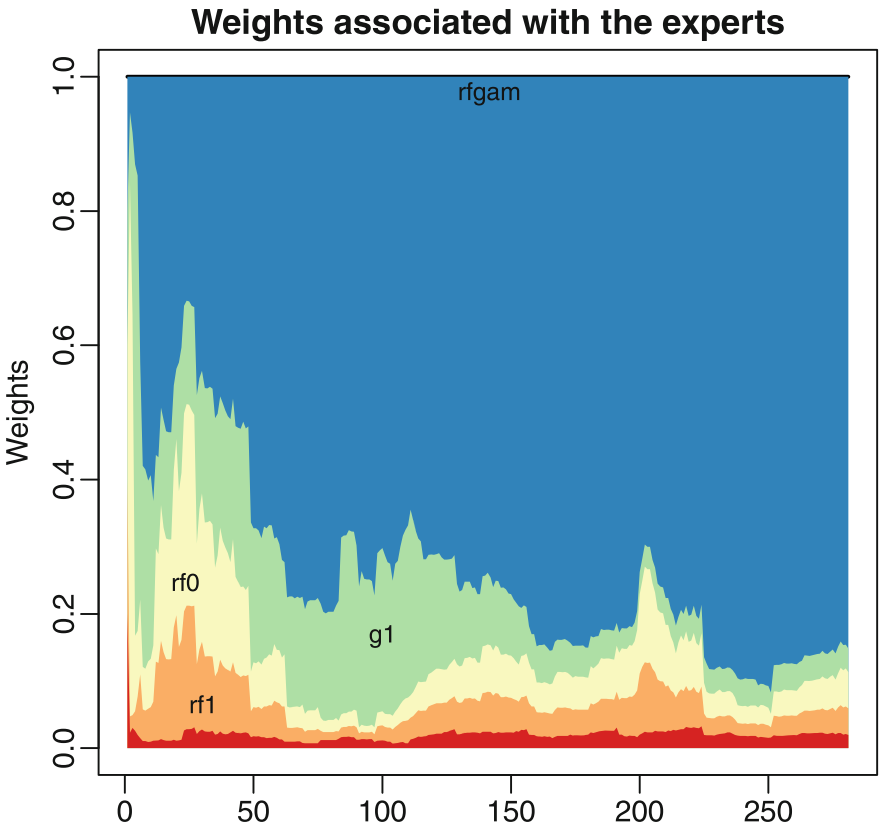


Fig. 4.6 Weights of the BOA algorithm on the test set

that the RF extrapolation is in the convex hull of the estimation data. This is not the case for the GAM models where we clearly see the linear negative extrapolation trend of `g0` and `g1`. Given that the effects of `g1` are added to `rfgam`, this RF can extrapolate as a GAM model. The ALE of `timeCount` on the aggregation is not linear, due the online adjustment of the weights. We also see that most of the time the aggregation trend falls in the convex hull of the other experts' trends.

Regarding the effect of `load48`, we see two groups of effects: the GAM ones (including `rfgam`), which are linear, and the RF ones which are close to piecewise linear functions. The ALE on the aggregation does not fall in the convex hull of the experts' ALEs, but it is closer to RF ones than to GAM ones. For the `temp` effect, all the models obtain similar effects for low temperatures (even if the GAM ones are again smoother than the RF ones), which is due to the impact of electrical heating. We observe some important differences for high temperatures between the models. This temperature effect has to be interpreted jointly with the `toy` effect, due to the yearly seasonality of the temperature. We observe very different yearly patterns, in particular during the summer holidays period when `g1` and `rfgam` clearly see a decrease of the load whereas the RFs and $g0$ effects are quite flat. Further, we see a gap in the RF effects between the beginning of the year and the end of the year, whereas the cyclic constraints included in the GAMs do not allow such a

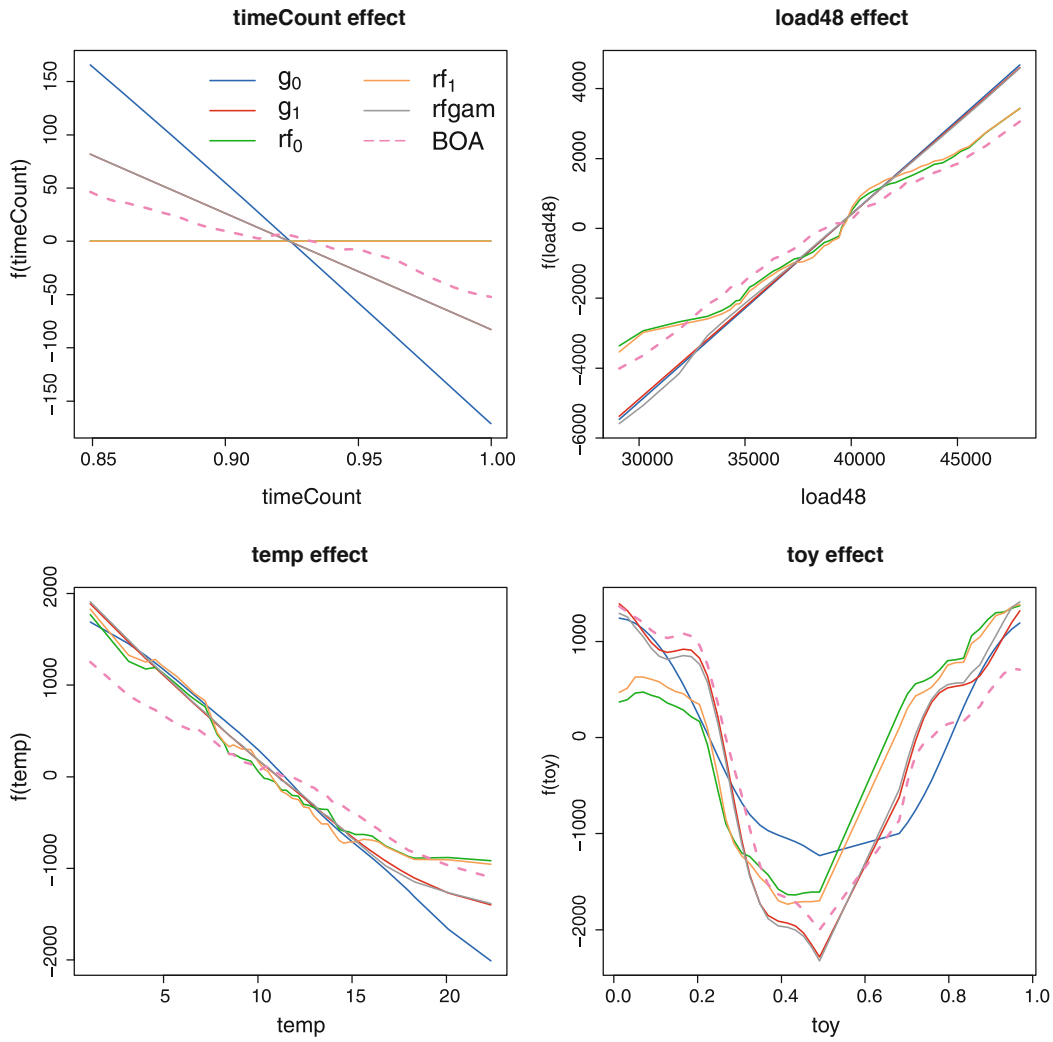


Fig. 4.7 Accumulated local effects for 4 covariates, for all our ML models and the online aggregation over the test set

discontinuity. Again, BOA produces effects similar but adapted to the test data for both temperature and yearly effects.

The ALE representation can be used also in an online setting, to visualize the time evolution of the different effects embedded into the aggregation. We plot these time varying ALEs for `temp` and `toy` in Fig. 4.8. We clearly see how the aggregation effects are learned sequentially. We also see that the domain of the effects (the x-axis range) evolves with time.

This experiment shows that we can adapt ALE plots to the context of online aggregation of experts and interpret it as an additive model, making it more explainable and thus increasing trust in the models. Another alternative could be to introduce additive constraints in the aggregation algorithm. This has been proposed by Capezza et al. [5], but not in an online setting.

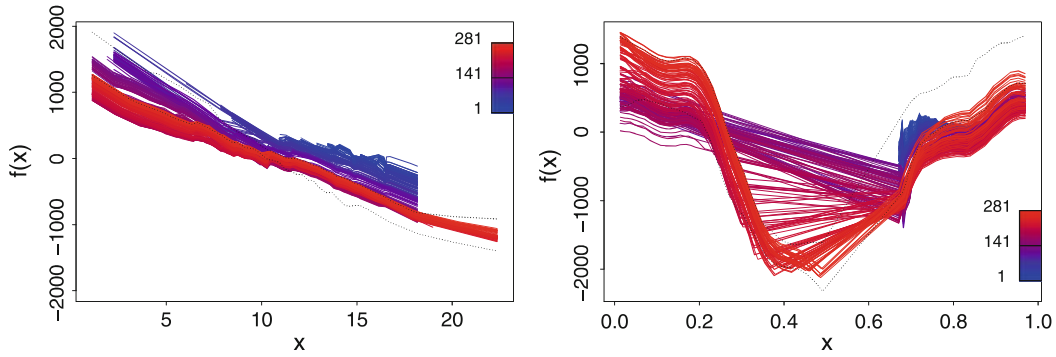


Fig. 4.8 Time varying accumulated local effects for temperature and time of year effects extracted from BOA aggregation (from the first instant of the test set in blue to final day in red). Dotted black lines correspond to the convex hull of the ALE effects of the individual experts

References

1. Apley DW, Zhu J (2020) Visualizing the effects of predictor variables in black box supervised learning models. J R Stat Soc Ser B (Stat Methodol) 82(4):1059–1086
2. Bourel M, Cugliari J, Goude Y, Poggi JM (2020) Boosting diversity in regression ensembles. https://hal.archives-ouvertes.fr/hal-03041309/
3. Breiman L (2001) Random forests. Mach Learn 45(1):5–32
4. Breiman L, Friedman J, Olshen R, Stone C (1984) Classification and regression trees. Chapman & Hall/CRC
5. Capezza C, Palumbo B, Goude Y, Wood SN, Fasiolo M (2021) Additive stacking for disaggregate electricity demand forecasting. Ann Appl Stat 15(2):727–746
6. Cesa-Bianchi N, Lugosi G (2006) Prediction, learning, and games. Cambridge University Press
7. Cesa-Bianchi N, Orabona F (2021) Online learning algorithms. Annu Rev Stat Appl 8(1):165–190
8. De Vilmarest J, Goude Y (2021) State-space models win the IEEE dataport competition on post-covid day-ahead electricity load forecasting. Tech. rep., arXiv:2110.00334
9. Duchon J (1977) Splines minimizing rotation-invariant semi-norms in Solobev spaces. In: Schemp W, Zeller K (eds) Construction theory of functions of several variables. Springer, Berlin, pp 85–100
10. Farrokhabadi M, Browell J, Wang Y, Makonin W, Zareipour H (2021) Day-ahead electricity demand forecasting competition: Post-covid paradigm. Tech. rep.
11. Fasiolo M, Nedellec R, Goude Y, Wood SN (2020) Scalable visualization methods for modern generalized additive models. J Comput Graph Stat 29(1):78–86
12. Flaxman S, Mishra S, Gandy A, Unwin HJT, Mellan TA, Coupland H, Whittaker C, Zhu H, Berah T, Eaton JW, et al (2020) Estimating the effects of non-pharmaceutical interventions on COVID-19 in Europe. Nature 584(7820):257–261
13. Gaillard P, Goude Y (2015) Forecasting electricity consumption by aggregating experts; how to design a good set of experts. In: Modeling and stochastic learning for forecasting in high dimensions. Springer, pp 95–115
14. Gaucher S, Goude Y, Antoniadis A (2021) Hierarchical transfer learning with applications for electricity load forecasting. Preprint. arXiv:211108512
15. Goehry B, Yan H, Goude Y, Massart P, Poggi JM (2021) Random forests for time series. REVSTAT Stat J. https://hal.archives-ouvertes.fr/hal-03129751/
16. Hastie T, Tibshirani R (1990) Generalized additive models. Chapman & Hall

17. Hastie T, Tibshirani R (1993) Varying-coefficient models. J R Stat Soc Ser B Methodol 55(4):757–796
18. IEA (2020) Year-on-year change in weekly electricity demand, weather corrected, in selected countries. https://www.iea.org/data-and-statistics/charts/year-on-year-change-in-weekly-electricity-demand-weather-corrected-in-selected-countries-january-december-2020
19. Jones M, Pewsey A (2009) Sinh-arcsinh distributions. Biometrika 96(4):761–780. https://doi.org/10.1093/biomet/asp054
20. Liu Z, Ciais P, Deng Z, Lei R, Davis SJ, Feng S, Zheng B, Cui D, Dou X, Zhu B, Guo R, Ke P, Sun T, Lu C, He P, Wang Y, Yue X, Wang Y, Lei Y, Zhou H, Cai Z, Wu Y, Guo R, Han T, Xue J, Boucher O, Boucher E, Chevallier F, Tanaka K, Wei Y, Zhong H, Kang C, Zhang N, Chen B, Xi F, Liu M, Bréon FM, Lu Y, Zhang Q, Guan D, Gong P, Kammen DM, He K, Schellnhuber HJ (2020) Near-real-time monitoring of global co2 emissions reveals the effects of the covid-19 pandemic. Nat Commun 11(1):5172. https://doi.org/10.1038/s41467-020-18922-7
21. Mayr A, Fenske N, Hofner B, Kneib T, Schmid M (2012) Generalized additive models for location, scale and shape for high dimensional data—a flexible approach based on boosting. J R Stat Soc Ser C (Appl Stat) 61(3):403–427
22. Meyer BH, Prescott B, Sheng XS (2021) The impact of the covid-19 pandemic on business expectations. Int J Forecasting. https://doi.org/10.1016/j.ijforecast.2021.02.009, https://www.sciencedirect.com/science/article/pii/S0169207021000509
23. Molnar C (2019) Interpretable machine learning. Lulu.com
24. Pullano G, Valdano E, Scarpa N, Rubrichi S, Colizza V (2020) Evaluating the effect of demographic factors, socioeconomic factors, and risk aversion on mobility during the covid-19 epidemic in France under lockdown: a population-based study. Lancet Digital Health 2(12):e638–e649
25. Pullano G, Valdano E, Scarpa N, Rubrichi S, Colizza V (2020) Population mobility reductions during covid-19 epidemic in France under lockdown. MedRxiv 29:2020
26. Reeve HW, Brown G (2018) Diversity and degrees of freedom in regression ensembles. Neurocomputing 298:55–68
27. Rigby RA, Stasinopoulos DM (2005) Generalized additive models for location, scale and shape. J R Stat Soc Ser C (Appl Stat) 54(3):507–554. https://doi.org/10.1111/j.1467-9876.2005.00510.x
28. Wintenberger O (2017) Optimal learning with Bernstein online aggregation. Mach Learn 106(1):119–141
29. Wood SN (2006) Low-rank scale-invariant tensor product smooths for generalized additive mixed models. Biometrics 62(4):1025–1036
30. Wood SN (2013) On p-values for smooth components of an extended generalized additive model. Biometrika 100(1):221–228
31. Wood SN (2013) A simple test for random effects in regression models. Biometrika 100(4):1005–1010
32. Wood SN (2017) Generalized additive models: an introduction with R, 2nd edn. CRC Press, Boca Raton
33. Wood SN (2021) Inferring UK COVID-19 fatal infection trajectories from daily mortality data: were infections already in decline before the UK lockdowns? Biometrics. https://doi.org/10.1111/biom.13462, https://onlinelibrary.wiley.com/doi/full/10.1111/biom.13462
34. Wood SN, Pya N, Säfken B (2016) Smoothing parameter and model selection for general smooth models (with discussion). J Am Stat Assoc 111:1548–1575
35. Wood SN, Li Z, Shaddick G, Augustin NH (2017) Generalized additive models for gigadata: Modeling the uk black smoke network daily data. J Am Stat Assoc 112(519):1199–1210. https://doi.org/10.1080/01621459.2016.1195744
36. Ziel F (2021) Smoothed bernstein online aggregation for day-ahead electricity demand forecasting. Tech. rep., arXiv:2107.06268