Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *doi: 10.18637/jss.v000.i00*

GeDS: An R Package for Regression, Generalized Additive Models and Functional Gradient Boosting, based on Geometrically Designed (GeD) Splines

Dimitrina S. Dimitrova
City St George's,
University of London

Vladimir K. Kaishev
City St George's,
University of London

Emilio L. Sáenz Guillén
City St George's,
University of London

Abstract

In recent years, geometrically designed variable knot splines, named GeDS, have emerged as a promising technique in the domain of spline regression, with Kaishev, Dimitrova, Haberman, and Verrall (2016) and Dimitrova, Kaishev, Lattuada, and Verrall (2023) showcasing their potential. In this paper, we introduce the R package **GeDS** that includes the implementation of two significant enhancements of the original GeDS methodology. The first broadens the applicability of GeDS to encompass generalized additive models (GAM), by implementing the *local scoring* algorithm using GeD splines as function smoothers. This approach stands as a competitive alternative, complementing existing practices suggested by Hastie and Tibshirani (1990) and Wood (2017), and implemented in the R packages **gam** and **mgcv**, respectively. Secondly, we incorporate functional gradient boosting (FGB) to estimate the number and location of the spline knots, as well as the associated regression coefficients. This novel approach allows the final boosted fit to be expressed as a single spline model, contrasting with typical gradient boosting models, which generally lack a straightforward, interpretable representation. We demonstrate that this technique yields competitive spline fits comparing favorably in both accuracy and efficiency to the outputs of existing boosting-with-splines procedures proposed by Bühlmann and Yu (2003) and Schmid and Hothorn (2008a), and implemented in the R package **mboost**.

The above extensions position GeDS as a versatile tool for additive modeling within the exponential family, suitable for both regression and classification tasks. The GeDS methodology, including GAM-GeDS and FGB-GeDS, is implemented in the R package **GeDS** available from `https://cran.r-project.org/package=GeDS`. We illustrate the capabilities of this package foregrounding the competitiveness of GeDS, and its potential for applications in the wider contexts of data science and machine learning.

Keywords: Variable-knot spline regression, Gradient Boosting, Generalized Additive Models.

1. Introduction

Geometrically designed spline (GeDS) regression with variable knots stands as a very competitive alternative to existing methods in the free-knot regression splines literature of recent years (see, e.g., Kaishev *et al.* (2016) for a brief review). The GeDS method is based on a residual-driven (locally-adaptive) knot insertion scheme that produces a piecewise linear spline fit, over which smoother higher order spline fits are subsequently built applying Schoenberg's variation diminishing approximation. GeDS was first introduced for the univariate Normal case by Kaishev *et al.* (2016). The authors demonstrate that GeDS estimation overcomes some major drawbacks of the existing knot optimization methods, namely "knot confounding" and "lethargy" problems (see Zhou and Shen (2001) and Jupp (1978)), without relying on costly non-linear optimization. It yields highly competitive outcomes utilizing a small number of knots for various signal-to-noise ratios, and is suitable for both sparse and dense data. Moreover, this is accomplished at a minimal computational cost, employing a stopping rule based on a ratio of consecutive deviances.

Dimitrova *et al.* (2023) have extended the GeDS methodology to the broader realm of generalized non-linear models (GNM)—that include generalized linear models (GLM) as a special case—in which the response variable may have any distribution from the exponential family. The authors conduct a comprehensive numerical examination that demonstrates how the advantageous features of the Normal GeDS methodology carry over into its extension to GNM/GLM models, favorably comparing with other existing spline methods. Furthermore, Dimitrova *et al.* (2023) introduce a multivariate extension of GeDS, by defining the predictor component of the GLM to be in the form of a multivariate tensor product spline function. The GeDS method is implemented in the R package **GeDS**, available from the Comprehensive R Archive Network (CRAN) at `http://CRAN.R-project.org/package=GeDS` (attracted over 30,000 downloads since released).

Standing out for its efficiency to produce highly competitive fits—namely, its ability to model intricate functions with minimal parameters (i.e., knots and regression coefficients)—the R implementation of the canonical GeDS methodology is, however, restricted to a predictor dimension of up to two covariates. Given this limitation and the complexity involved in extending tensor product splines beyond two dimensions, we have identified two major strands in the literature where extending the GeDS methodology and enhancing its software implementation would be particularly impactful: generalized additive models and functional gradient boosting.

On the one hand, generalized additive models (GAM; Hastie and Tibshirani (1986, 1990)) provide a flexible statistical modeling technique that extends generalized linear models (GLM) to allow for the modeling of predictor effects via non-linear *smooth functions* of the features. This enables to capture more complex patterns while retaining the interpretability inherent to GLM. Hastie and Tibshirani's method to fit GAM employs the *local scoring* and *backfitting* algorithms, in conjunction with scatterplot smoothers for the fitting of individual functions. Their approach is implemented in the R package **gam** (Hastie (2024)), which currently supports local regression and smoothing splines to fit the models' smooth functions. Wood (2017) proposes instead a penalized regression spline approach, with automatic smoothness selection. The latter is implemented in the **mgcv** R package (Wood (2023)) which extends **gam** allowing for more general smoothers, including two-dimensional smoothers for spatial data or tensor product smoothers for interactions.

On the other hand, functional gradient boosting (FGB) constitutes a pivotal and widely adopted methodology in machine learning, particularly well-suited for regression and classification tasks in high-dimensional data settings. The original boosting procedure emerged from the field of machine learning and was first proposed by Schapire (1990), who discussed that "arbitrarily high accuracy could be achieved in an algorithm utilizing 'weak learners'". This embryonic idea laid the basis for the development of the widely recognized Adaptive Boosting (AdaBoost) classification algorithm, introduced by Freund and Schapire (1996, 1997). Breiman (1998, 1999) later demonstrated that AdaBoost could be interpreted as a gradient descent algorithm with a particular loss function. This perspective was furthered by Friedman, Hastie, and Tibshirani (2000) and Friedman (2001), who tailored the boosting concept to the field of statistical modeling and developed the first explicit regression gradient boosting algorithms. In the aftermath of these foundational works, numerous statistical boosting algorithms have been proposed in the academic literature (see, for example, Mayr, Binder, Gefeller, and Schmid (2014) for an overview). At its core, boosting constitutes an ensemble machine learning technique aimed at building a single "strong" learner by iteratively combining multiple simple models (or "weak learners"), each striving to enhance the performance of the preceding accumulative model. A particularly relevant boosting algorithm is component-wise (or model-based) gradient boosting (Bühlmann and Yu (2003), Bühlmann and Hothorn (2007)). The latter was introduced as a competitive alternative to standard estimation techniques for (generalized) additive models, such as backfitting. Unlike traditional methods for fitting additive models, component-wise boosting inherently performs variable selection, which makes it especially suitable for high-dimensional problems. This approach has more recently been extended by Hofner, Mayr, and Schmid (2016) to allow for the simultaneous modeling of multiple distribution parameters.

While regression trees are the most popular base procedure in boosting, splines have also been frequently considered. For instance, Bühlmann and Yu (2003) examined the L_2Boost algorithm in detail and determined that L_2Boost using smoothing splines as learners achieves optimal minimax rates of convergence in both regression and classification. The authors present L_2Boosting in conjunction with component-wise cubic smoothing splines as a "practical and efficient" procedure, particularly when handling high-dimensional predictors. They claim that their approach outperforms L_2Boost with stumps (i.e., a tree with two terminal nodes) and other traditional competitors. Schmid and Hothorn (2008a) consider fitting additive regression models using L_2Boost, but with P-splines functions of the predictors instead. These yield similar prediction errors to smoothing splines, but are more advantageous from a computational perspective. Boosting using component-wise P-splines is implemented in the **mboost** R package (Hothorn, Buehlmann, Kneib, Schmid, and Hofner (2022)).

Both in the case of GAM as well as for boosting, the above mentioned spline-focused designs require the pre-determination by the user of the number of knots, which are then positioned equidistantly. In addition, the degree of smoothness of the spline is ruled globally by a sole data-driven penalty parameter, thereby eliminating the possibility of any local adjustment. Yet, as demonstrated by Dimitrova *et al.* (2023), there are many applications where an elevated level of adaptability within the spline predictor component, allowing for local smoothness regulation, is sought.

In this context, geometrically designed (GeD) variable knot regression splines represent a compelling alternative. First, we extend GeDS to accommodate the family of generalized additive spline models, integrating GeD splines as function smoothers at each backfitting iteration within the local scoring algorithm. Second, we introduce a novel functional gradient boosting algorithm that employs GeD splines as base-learners. Both GAM-GeDS and FGB-GeDS propel GeDS into the additive modelling framework within the exponential family of distributions. On the one hand, GAM-based extensions may be advantageous when the interpretability of models, visualization of effects, and flexibility in representing diverse relationships are paramount. On the other hand, FGB-based extensions may be more suited to scenarios where superior predictive performance and robustness are essential, making it apt for handling complex, high-dimensional data.

The above extensions result in highly competitive spline fitting that surpasses similar competing approaches both in accuracy and efficiency. At the same time, they improve the flexibility and adaptability of the GeDS methodology, thereby broadening the scope of its applicability to more complex and diverse problem domains.

Another remarkable hallmark is that the ability to express GeDS base-learners as piecewise polynomial functions enables the representation of the FGB-GeDS model as a single spline model. This contrasts with typical gradient boosting models—employing, for example, trees or smoothing splines/P-splines—which generally lack a compact, interpretable representation, and are often referred to as "black box" models. In addition, FGB-GeDS efficiently addresses the selection of an optimal number of boosting iterations through a stopping rule based on a ratio of consecutive deviances, thus avoiding time-consuming techniques like cross-validation.

In this paper we describe the statistical framework underlying the GAM-GeDS and FGB-GeDS methodologies. Additionally, we introduce the R package **GeDS**, in which these two latest extensions are implemented, along with the canonical GeDS technique. We illustrate the usage of **GeDS** and highlight its advantages when compared to the approaches from the **gam**, **mgcv**, and **mboost** packages. The structure of the paper is as follows. In Section 2, we briefly describe the GeDS methodology. In Section 3, generalized additive models with GeD splines is presented. In Section 4, we outline the basic notions of functional gradient boosting for regression and classification, and in Section 5 the FGB-GeDS algorithm is introduced. In Sections 6, 7 and 8, we thoroughly study the numerical properties of GAM-GeDS and FGB-GeDS and compare them with the **gam** and **mgcv** packages estimators, on the one hand, and the **mboost** package estimators, on the other. Finally, in Section 9, we provide some conclusions and discuss further possible extensions of the **GeDS** package and methodology.

2. GeDS estimation method

We start by describing the original univariate Normal GeDS estimation method presented in Kaishev *et al.* (2016), which is implemented as `NGeDS()` in the **GeDS** R package. Consider a response variable Y and a sole independent variable X, with $X \in [a, b]$, $a, b \in \mathbb{R}$, and assume there is a relationship between X and Y of the form:

$$Y = f(X) + \epsilon \tag{1}$$

where $f(\cdot)$ is an unknown function and ϵ is a random (normal) error variable with zero mean, $\mathbb{E}[\epsilon] = 0$, and constant variance, $\mathbb{E}[\epsilon^2] = \sigma_\epsilon^2$. A possible solution to the regression problem of estimating $f(\cdot)$ based on a sample of observations $\{Y_i, X_i\}_{i=1}^N$, is to approximate f with an n-th order (i.e., degree $n-1$) spline function on $[a, b]$.

More specifically, denote by $S_{\boldsymbol{t}_{\kappa,n}}$ the linear space of all n-th order spline functions defined on a set of non-decreasing knots $\boldsymbol{t}_{\kappa,n} = \{t_i\}_{i=1}^{2n+\kappa}$, where $t_n = a$, $t_{n+\kappa+1} = b$. We consider splines with simple knots, except for the n left and right most knots which will be assumed coalescent, i.e. $\boldsymbol{t}_{\kappa,n} = \{t_1 = ... = t_n < t_{n+1} < ... < t_{n+\kappa} < t_{n+\kappa+1} = ... = t_{2n+\kappa}\}$. By the Curry-Schoenberg theorem, a spline regression function $f \in S_{\boldsymbol{t}_{\kappa,n}}$, can be expressed as

$$f(\boldsymbol{t}_{\kappa,n}; x) = \boldsymbol{\theta}' \boldsymbol{N}_n(x) = \sum_{i=1}^{p} \theta_i N_{i,n}(x) \tag{2}$$

where $\boldsymbol{\theta} = (\theta_1, ..., \theta_p)'$ is a vector of real valued regression coefficients and $\boldsymbol{N}_n(x) = (N_{1,n}(x), ..., N_{p,n}(x))'$, $p = n + \kappa$, are B-splines of order n, defined on $\boldsymbol{t}_{\kappa,n}$. It is well known that $\sum_{i=j-n+1}^{j} N_{i,n}(t) = 1$ for any $t \in [t_j, t_{j+1})$, $j = n, ..., n+\kappa$ and $N_{i,n}(t) = 0$ for $t \notin [t_i, t_{i+n}]$. Thus, for a fixed spline order n and given a sample $\{Y_i, X_i\}_{i=1}^{N}$, the spline regression problem boils down to estimating the number of (internal) knots κ, their locations $\boldsymbol{t}_{\kappa,n}$, and the regression coefficients $\boldsymbol{\theta}$.

In this regard, geometrically designed splines (GeDS) are introduced as a novel variable knot spline regression estimation technique. This method is inspired by an innovative geometric interpretation of parameter estimation in spline regression as akin to pinpointing "control points". Indeed, the spline regression function can be regarded as a special case of a parametric spline curve, characterized by a corresponding "control polygon". In computer graphics and geometric design, a control polygon consists of a sequence of connected nodes (control points) in space, that is used to define and manipulate an object's shape. By adjusting these control points, rather than the spline points, we can make localized and precise alterations to the curve, offering greater control over the spline's form.

The interpretation of the spline regression function as a parametric spline curve, defined and shaped by a control polygon, suggests that given n and κ—i.e., the order of the spline and the number of internal knots—determining the knot positions and regression coefficients of the spline regression function is tantamount to locating the x and y-coordinates of the vertices of the control polygon of the parametric spline curve. Therefore, and since the control polygon itself is a linear spline function, GeDS starts by constructing a control polygon as a linear spline fit to the data. Due to the shape-preserving and convex hull properties that a spline curve holds with respect to its control polygon, the geometric position of this control polygon defines then the location of the higher order, smoother spline regression functions that are subsequently built.

The GeDS method unfolds into two phases. In stage A, a least squares linear spline fit to the data is constructed. This is viewed as the initial position of the control polygon of a higher order spline regression curve. In stage B, a higher order spline function—designed to reduce variations and provide a least-squares fit to the data—is used to approximate the fitted polygon from Stage A. Indeed, this roughly coincides with the Schoenberg's variation-diminishing spline approximation of the control polygon. In a similar way, designers in Computer Aided Geometric Design applications construct a control polygon to capture the shape of the curve underlying some noisy data, and then compute smoother higher order Schoenberg's variation diminishing spline curves that closely follow the initial control polygon and hence the desired shape. Stage A and stage B of GeDS are briefly summarized as follows; for a more detailed description, see Kaishev *et al.* (2016).

Stage A

Stage A of a GeDS model is dedicated to finding the optimal linear spline fit to the data, i.e., the control polygon that best captures the underlying functional shape determined by the data. In the Normal case, we start with a straight line least-squares (LS) fit to the data. This fit is then sequentially "broken" into a piecewise linear LS fit, by iteratively introducing knots at those points where the fit most deviates from the underlying functional shape determined by the data, according to a bias driven measure that is computed across appropriately defined clusters of residuals (see Section 3 in Kaishev *et al.* (2016)). The resulting LS linear spline fit is denoted by $\hat{f}\left(\boldsymbol{\delta}_{\kappa,2}; \hat{\boldsymbol{\alpha}}_p; x\right) = \sum_{i=1}^{p} \hat{\alpha}_i N_{i,2}(x)$ with number of internal knots κ, number of B-splines $p = \kappa + 2$ and knots locations $\boldsymbol{\delta}_{\kappa,2} = \{\delta_1 = \delta_2 < \delta_3 < \ldots < \delta_{\kappa+2} < \delta_{\kappa+3} = \delta_{\kappa+4}\}$. The knot insertion stops when adding more knots does not significantly improve the fit according to the following residual sum of squares (RSS) criterion:

$$\mathrm{RSS}(\kappa+q)/\mathrm{RSS}(\kappa) = \sum_{i=1}^{N}\left(y_i - \hat{f}\left(\boldsymbol{\delta}_{\kappa+q,2}; x_i\right)\right)^2 \Bigg/ \sum_{j=1}^{N}\left(y_i - \hat{f}\left(\boldsymbol{\delta}_{\kappa,2}; x_i\right)\right)^2 \geq \phi_{exit} \qquad (3)$$

where $q \geq 1$ and $\phi_{exit} \in (0,1)$ is a certain pre-specified threshold level close to one. If the inequality in (3) is satisfied, it implies that the fit $\hat{f}(\boldsymbol{\delta}_{\kappa,2}; \hat{\boldsymbol{\alpha}}_p; x) = \sum_{i=1}^{p} \hat{\alpha}_i N_{i,2}(x)$ does not significantly improve if q more knots are added to the model. Therefore, $\hat{f}(\boldsymbol{\delta}_{\kappa,2}; \hat{\boldsymbol{\alpha}}_p; x)$ is the selected linear spline model which reproduces the shape of the unknown, underlying function f. Equation (3) thus serves both as a stopping rule and model selector.

Stage A of Normal GeDS is extended to the more general GNM/GLM context by replacing LS fitting by Iteratively Reweighted Least Squares (IRLS) fitting and the stopping rule in (3) by a deviance-based stopping criterion (see Dimitrova *et al.* (2023)). Hence, analogously, starting from a straight line fit and adding one knot at a time, we follow the IRLS procedure to find the linear spline fit $\hat{f}(\boldsymbol{\delta}_{\kappa,2}; \hat{\boldsymbol{\alpha}}_p; x)$ such that, for $q \geq 1$,

$$\frac{D(\hat{\boldsymbol{\alpha}}_{p+q}; \kappa+q; 2)}{D(\hat{\boldsymbol{\alpha}}_p; \kappa; 2)} \geq \phi_{exit}, \qquad (4)$$

which is a direct generalization of (3), based on the deviance $D(\hat{\boldsymbol{\alpha}}_{p+q}; \kappa+q; 2)$. Testing the inequality in (4) serves as the stage A model selector: if the number of internal knots κ, is such that the inequality in (4) is fulfilled for the first time in the knot addition process, then it means that $\hat{f}(\boldsymbol{\delta}_{\kappa+q,2}; \hat{\boldsymbol{\alpha}}_{p+q}; x)$ does not significantly improve with the inclusion of the last q additional knots, and therefore, $\hat{f}(\boldsymbol{\delta}_{\kappa,2}; \hat{\boldsymbol{\alpha}}_p; x)$ is the selected model that captures the shape of the underlying data at the predictor scale of the GNM/GLM.

Stage B.1.

Given the (final) linear fit $\hat{f}\left(\boldsymbol{\delta}_{\kappa,2}, \hat{\boldsymbol{\alpha}}_p; x\right)$ from stage A, with κ internal knots, the set of knots $\bar{\boldsymbol{t}}_{\kappa-(n-2),n}$ (with $\kappa-(n-2)$ internal knots) for each order $n = 3, \ldots, n_{\max}$ is obtained by averaging the knots in the set $\boldsymbol{\delta}_{\kappa,2}$ as follows:

$$\bar{t}_{i+n} = \left(\delta_{i+2} + \ldots + \delta_{i+n}\right) \Big/ (n-1), \quad i = 1, \ldots, \kappa-(n-2). \qquad (5)$$

As shown by Kaishev *et al.* (2016), by choosing the knots $\bar{\boldsymbol{t}}_{\kappa-(n-2),n}$ according to (5), the n-th order spline predictor curve $f\left(\bar{\boldsymbol{t}}_{\kappa-(n-2),n}, \hat{\boldsymbol{\alpha}}_p; x\right)$ becomes nearly the Schoenberg variation diminishing spline (VDS) approximation to the linear fit, $\hat{f}\left(\boldsymbol{\delta}_{\kappa,2}, \hat{\boldsymbol{\alpha}}_p; x\right)$, from stage

A. Consequently, the linear fit $\hat{f}\left(\boldsymbol{\delta}_{\kappa,2}, \hat{\boldsymbol{\alpha}}_p; x\right)$ can be viewed as (nearly) the control polygon of the predictor curve $f\left(\bar{\boldsymbol{t}}_{\kappa-(n-2),n}, \hat{\boldsymbol{\alpha}}_p; x\right)$. Error bounds for this VDS approximation and optimality properties of the knots are derived and discussed in Kaishev *et al.* (2016).

The spline predictor curve $f(\bar{\boldsymbol{t}}_{\kappa-(n-2),n}, \hat{\boldsymbol{\alpha}}_p; x)$ obtained at stage B.1, closely follows the shape of $\hat{f}(\boldsymbol{\delta}_{\kappa,2}, \hat{\boldsymbol{\alpha}}_p; x)$ and, thus, of the data. However, it is not formally a maximum likelihood estimate of the data $\{Y_i, X_i\}_{i=1}^N$. To rectify this, stage B.2 treats the B-spline coefficients $\hat{\boldsymbol{\alpha}}_p$ as unknown parameters (now denoted by $\boldsymbol{\theta}_p$, $p = \kappa + 2,$), and re-estimates them in a final run of the LS/IRLS procedure, while preserving the same set of knots $\bar{\boldsymbol{t}}_{\kappa-(n-2),n}$.

Stage B.2

For each fixed order $n = 3, ..., n_{\max}$, find the maximum likelihood estimates $\hat{\boldsymbol{\theta}}_p$ of the B-spline coefficients, $\boldsymbol{\theta}_p$, of the spline predictor curve $f(\bar{\boldsymbol{t}}_{\kappa-(n-2),n}, \hat{\boldsymbol{\theta}}_p; x)$ with knots determined in stage B.1. Among all fits $\hat{f}(\bar{\boldsymbol{t}}_{\kappa-(n-2),n}, \hat{\boldsymbol{\theta}}_p; x)$, of order $n = 2, ..., n_{\max}$—i.e. including the linear fit, $\hat{f}(\boldsymbol{\delta}_{\kappa,2}, \hat{\boldsymbol{\alpha}}_p; x)$ from stage A—choose the one of order $\hat{n}$, for which the deviance is minimal. In this way, in addition to the number and location of the internal knots, GeDS also estimates the degree of the spline. Of course, any of the produced final fits of order $n \neq \hat{n}$ can be used if other features are more desirable, for example, if better smoothness is required.

Package **GeDS** provides an R implementation of geometrically designed spline (GeDS) regression. Package **GeDS** can be installed and loaded in an R session via:

```
R> install.packages("GeDS")
R> library(GeDS)
```

The functions implementing GeDS regression are `NGeDS()` and `GGeDS()`. On the one hand, `NGeDS()` function constructs a geometrically designed (univariate or bivariate) variable knot spline regression model, for a response having a Normal distribution. The synopsis of this function is:

```
NGeDS(formula, data, weights, beta = 0.5, phi = 0.99,
min.intknots = 0, max.intknots = 500, q = 2, Xextr = NULL, Yextr = NULL,
show.iters = FALSE, stoptype = "RD", higher_order = TRUE,
intknots_init = NULL, fit_init = NULL, only_pred = FALSE)
```

On the other hand, `GGeDS()` constructs a geometrically designed (univariate or bivariate) variable knot spline regression model for the predictor term of a generalized (non-)linear model, where the response follows a pre-specified distribution from the exponential family. The synopsis of this function is:

```
GGeDS(formula, family = gaussian(), data, weights, beta, phi = 0.99,
min.intknots, max.intknots, q = 2L, Xextr = NULL, Yextr = NULL,
show.iters = FALSE, stoptype = "SR", higher_order = TRUE)
```

A GeDS model is specified using a `formula` of the form `Y ~ f(X)` or `Z ~ f(X,Y)`. If needed, prior weights on observations can be assigned through the `weights` vector. Arguments `beta` and `phi` are numeric parameters in the interval $[0, 1]$. While `beta` tunes the knot placement in stage A of GeDS—via the aforementioned bias driven measure of appropriately defined clusters of residuals (see Kaishev *et al.* (2016) for details on this parameter)—, `phi` specifies the threshold for the stopping rule in stage A of GeDS and `q` is an integer for fine-tuning the latter rule (see (3) and (4)). Different stopping rules beyond (3) and (4) can be chosen through `stoptype`, see the package documentation for details. `Xextr` and `Yextr` are numeric vectors of 2 elements representing the left-most and right-most limits of the intervals embedding the observations of the first and second (if bivariate GeDS is run) independent variables, respectively. Finally, in `GGeDS()`, the `family` argument determines the *link function* to be used in the GNM/GLM.

3. Generalized additive models with GeDS

In this section, we introduce generalized additive models with GeD splines which, as demonstrated by the examples in Section 7, extend the applicability of GeDS to truly multivariate settings. Additive models (AM) provide a useful extension of linear models, making them more flexible while still retaining much of their interpretability (see, e.g., Hastie, Tibshirani, Friedman, and Friedman (2009)). Specifically, AMs allow for the incorporation of nonlinear smooth functions of the covariates and, unlike other generalizations (e.g., surface smoothers), they maintain an additive structure that permits the separate analysis of the predictor effects. An additive model takes the form

$$\mathbb{E}\left[Y|X^1,...,X^P\right] = \alpha + \sum_{j=1}^{P} f^j(X^j) \qquad \text{or} \qquad Y = \alpha + \sum_{j=1}^{P} f^j(X^j) + \varepsilon \tag{6}$$

where the error term ε is assumed to have zero mean, $\mathbb{E}[\varepsilon] = 0$, constant variance, $\mathbb{E}[\varepsilon^2] = \sigma^2$, and to be independent of the predictor variables $X^1,...,X^P$. It is also implicitly assumed that $\mathbb{E}[f^j(X^j)] = 0$ (which implies $\mathbb{E}[Y] = \alpha$), since otherwise there would be unaccounted constants in each of the functions f^j (see Hastie and Tibshirani (1990)). For simplicity, we will take f^j to be arbitrary univariate functions, although, as shown in Example 7.1, it is also possible to include functions of two or more dimensions as model components.

The additive model is a valuable data-analytic tool that, like the linear model, expresses the response as a sum of functions of individual predictors. This formulation enables the visualization of each predictor's contribution to the predicted response after model fitting. Although the additive model is typically an approximation of the true regression surface, the goal is for this approximation to be accurate enough to identify significant predictors and elucidate their effects (see Hastie and Tibshirani (1990)).

Various techniques can be used to estimate additive models. In this context, the *backfitting algorithm* provides a flexible alternative, allowing the fitting of an additive model using any regression-based fitting mechanism. The rationale behind this algorithm is intuitive: if the additive model specified in equation (6) is correct then for any k, $\mathbb{E}\left[Y - \alpha - \sum_{j\neq k} f^j(X^j)|X^k\right] = f^k(X^k)$ must hold. This relationship suggests an iterative method for estimating the functions f^j, which we present in terms of arbitrary scatterplot smoothers S^j in Algorithm 1 (see Hastie *et al.* (2009)).

Generalized additive models broaden the scope of additive models by allowing the response variable to be assumed to follow any distribution from the exponential family. This specifically implies that, in the modeling process, the mean of the response is linked to the predictors through a general *link function*, which is associated with the assumed distribution of the

Algorithm 1 The Backfitting Algorithm for Additive Models

1. Initialize: $\hat{\alpha} = \frac{1}{N}\sum_{i=1}^{N} Y_i$, $\hat{f}^j = 0, \forall j$
2. For each base-learner $\hat{f}^j$, $j = 1, ..., P$:

$$\hat{f}^j \leftarrow S^j \left[\left\{ Y_i - \hat{\alpha} - \sum_{k \neq j} \hat{f}^k \left(X_i^k\right) \right\}_{i=1}^{N} \right]$$

$$\hat{f}^j \leftarrow \hat{f}^j - \frac{1}{N}\sum_{i=1}^{N} \hat{f}^j(X_i^j)$$

3. Repeat Step 2 until

$$\text{RSS} = \sum_{i=1}^{N} \left(Y_i - \hat{\alpha} - \sum_{j=1}^{P} \hat{f}^j(X_i^j) \right)^2$$

fails to decrease.

response. In other words, GAMs extend generalized linear models by replacing the linear predictor, $\alpha + \sum_j X^j \beta^j$, with $\alpha + \sum_j f^j(X^j)$, where the f^j represent smooth functions of the predictor variables X^j. This is analogous to how additive models extend linear regression models by incorporating non-linear functions of the predictors.

Therefore, in generalized additive models, the response variable Y is assumed to follow a distribution from the exponential family with mean $\mu = \mathbb{E}\left[Y|X^1, ..., X^P\right]$. The model then relates μ to the predictor variables $X^1, ..., X^P$ via a link function $g(\cdot)$:

$$g(\mu) = \alpha + \sum_{j=1}^{P} f^j(X^j), \qquad \mathbb{E}\left[f^j(X^j)\right] = 0, \quad j = 1, ..., P \tag{7}$$

Once the distribution of the response is chosen, an appropriate link function $g(\cdot)$ is selected. This function determines the way the mean response μ is transformed to the additive predictor scale, thereby defining the form of the adjusted dependent variable $\boldsymbol{z}$ and the iteration-specific weights $\boldsymbol{w}$ (see Algorithm 2). Given these quantities, estimation of α and $f^1, ..., f^P$ is undertaken through the so-called *local scoring algorithm* as detailed in Algorithm 2 (see, e.g., SAS Institute Inc. (2018)). Table 1 displays $g(\mu)$, $\boldsymbol{z}$ and $\boldsymbol{w}$ for some common models in the context of GAM.

Some asymptotic theory has been developed for backfitting and local scoring estimators. Early work by Buja, Hastie, and Tibshirani (1989) established conditions for consistency and nondegeneracy, and proved convergence of backfitting for a class of smoothers, including cubic spline smoothers. Opsomer (2000) extended this by deriving recursive expressions for the asymptotic bias and variance of backfitting estimators based on local polynomial regression smoothers, and demonstrates that these estimators achieve the same convergence rate as univariate local polynomial regression. Additionally, this author provides explicit expressions for asymptotic bias and variance, along with optimal bandwidth parameters, in

Distribution	Link Function, $g(\mu)$	Adjusted Dependent Variable, $\boldsymbol{z}$	Weights, $\boldsymbol{w}$
Normal	Identity: μ	y	1
Binomial$(m,\mu)/m$	Logit: $\log\left(\frac{\mu}{1-\mu}\right)$	$\eta+\frac{y-\mu}{\mu(1-\mu)}$	$m\mu(1-\mu)$
Gamma	Inverse: $1/\mu$	$\eta-(y-\mu)/\mu^2$	μ^2
Poisson	Log: $\log(\mu)$	$\eta+(y-\mu)/\mu$	μ
Inverse Gaussian	Inverse squared: $1/\mu^2$	$\eta-2(y-\mu)/\mu^3$	μ^3

Table 1: Link function, $g(\cdot)$, adjusted dependent variable, $\boldsymbol{z}$ and weights, $\boldsymbol{w}$, for some commonly used models.

Algorithm 2 The General Local Scoring Algorithm

1. **Initialize:** $\hat{\alpha}=g\left(\frac{1}{N}\sum_{i=1}^N Y_i\right)$, $\hat{f}_m^j=0$, $j=1,...,P$, and $m=0$.
2. **Iterate:** Set $m=m+1$ and iterate to form the predictor $\hat{\boldsymbol{\eta}}$, the mean $\hat{\boldsymbol{\mu}}$, the weights $\boldsymbol{w}$, and the adjusted dependent variable $\boldsymbol{z}$:
   (a) Form the adjusted dependent variable
   $$z_i=\hat{\eta}_i^{m-1}+\left(Y_i-\hat{\mu}_i^{m-1}\right)\cdot\left(\frac{\partial\hat{\eta}}{\partial\hat{\mu}}\right)_i^{m-1},$$
   where $\hat{\eta}_i^{m-1}=\hat{\alpha}+\sum_{j=1}^P\hat{f}_{m-1}^j(X_i^j)$, $\hat{\mu}_i^{m-1}=g^{-1}\left(\hat{\eta}_i^{m-1}\right)$ and $i=1,...N$.
   (b) Form the weights: $w_i=\left(V_i^{m-1}\right)^{-1}\cdot\left[\left(\frac{\partial\hat{\mu}}{\partial\hat{\eta}}\right)_i^{m-1}\right]^2$, where V_i^{m-1} is the variance of Y at $\hat{\mu}_i^{m-1}$.
   (c) Fit an additive model to $\boldsymbol{z}$ by using the backfitting algorithm with weights $\boldsymbol{w}$ to obtain the estimated functions $\hat{f}_m^j(\cdot)$, $j=1,...P$, and the model $\hat{\boldsymbol{\eta}}^{\boldsymbol{m}}$.
3. **Until:** The empirical deviance $\sum_{i=1}^N \operatorname{dev}\left(Y_i,\hat{\mu}_i^m\right)$ fails to decrease.

the case of independence between covariates. In the same vein, Kauermann and Opsomer (2003) showed that the local scoring estimator inherits the asymptotic properties of some consistent local likelihood estimator for generalized additive models that, provided certain uniqueness conditions are met, attains the same asymptotic convergence rates as univariate local polynomial regression estimators.

Our GAM-GeDS implementation involves applying the local scoring algorithm (Algorithm 2), using Normal GeD splines as the function smoothers, S^j, within the backfitting algorithm (Algorithm 1). The function `NGeDSgam`, which applies this method, is as follows:

```
NGeDSgam(formula, family = "gaussian", data, weights = NULL, offset = NULL,
normalize_data = FALSE, min_iterations, max_iterations,
phi_gam_exit = 0.99, q_gam = 2, beta = 0.5, phi = 0.99, internal_knots = 500,
q = 2, higher_order = TRUE)
```

Now the model is specified using a formula of the type `Y~ f(X_1)+f(X_2)+...` and data should be provided as `data.frame` via the `data` argument. Data can be standardized before fitting the model by setting `normalize_data = TRUE`, and a minimum and maximum number of local scoring iterations can be set through `min_iterations` and `max_iterations`; `phi_gam_exit` and `q_gam` are the tuning parameters for the stopping rule of the local scoring iterations in Step 3 of Algorithm 2, following (4). Finally, `beta`, `phi`, `internal_knots` and `q` are parameters tuning the GeDS function smoothers, f^j, at each backfitting iteration.

4. Functional gradient boosting

In this section, we turn our attention to functional gradient boosting, another powerful technique in statistical learning, with the aim of incorporating GeD splines as base-learners (as detailed in Section 5). We begin by introducing the fundamental boosting framework, and then briefly explore its application in the context of regression, classification, and, more generally, the exponential family.

Friedman (2001) demonstrates that boosting algorithms can be regarded as an empirical risk optimization technique implementing steepest gradient descent in function space. Consider the i.i.d. random variables (Y, X) where Y is a one-dimensional response (or output) variable and X is a P-dimensional vector of explanatory (or input) variables, which we also refer to as covariates. The objective is to estimate the optimal prediction function

$$F^*(\cdot) = \underset{F(\cdot)}{\arg\min}\, \mathbb{E}\left[L(Y, F(X)\right] \tag{8}$$

that maps X to Y, where $F : \mathbb{R}^P \mapsto \mathbb{R}$ and $L(\cdot,\cdot) : \mathbb{R} \times \mathbb{R} \mapsto \mathbb{R}^+$. In other words, $F^*(\cdot)$ is defined to be the population minimizer of a given loss function $L(\cdot)$ over the joint distribution of (Y, X) (see Friedman *et al.* (2000)). To allow for gradient descent optimization of the loss and ensure convergence to a sole global minimum, $L(\cdot)$ is usually assumed to be differentiable and convex with respect to $F(\cdot)$.

Given a learning sample of observations $(Y_1, X_1), \ldots, (Y_N, X_N)$, estimation of $F^*(\cdot)$ is then undertaken by minimizing the empirical risk via iterative gradient descent in function space, that is, $\widehat{F}^*(\cdot) = \arg\min_{F(\cdot)} \frac{1}{N} \sum_{i=1}^{N} L\left(Y_i, F(X_i)\right)$. Functional gradient boosting, as given by Friedman (2001), is presented in Algorithm 3.

Choosing an appropriate value for the stopping iteration m_{stop} is necessary to prevent FGB from overfitting the data sample. This can be done by means of an appropriate "stopping rule". For example, Mayr, Hofner, and Schmid (2012) propose an "earlier stopping" approach that depends on AIC-based information criteria, while Bühlmann and Van De Geer (2011) suggest that m_{stop} can be determined via cross-validation.

Furthermore, the step length parameter ν, acting as a shrinkage factor for the gradient estimates at each boosting iteration, must be determined. According to Friedman (2001), the step length should be obtained by minimizing, at each boosting iteration, the objective function value (see step 2.iii in Algorithm 3). This methodological choice explains why Friedman's approach is often termed as "steepest descent", which can be seen as a particular variant of gradient descent. In contrast, Bühlmann and Van De Geer (2011) consider the choice of ν to be of minor importance as long as it is "small" enough to allow for gradual learning and reduce the risk of overfitting. All in all, the ν/m_{stop} trade-off is evident: smaller values of ν give rise to larger optimal m_{stop}-values, and viceversa (see Friedman (2001)).

Algorithm 3 Functional Gradient Boosting

Given data $\{Y_i, X_i\}_{i=1}^N$, a differentiable loss function $L\left(Y, F(X)\right)$ and a stopping number of iterations m_{stop}, the generic FGB algorithm proceeds as follows:

1. Initialize the model with a constant value (initial learner). A common choice is the empirical risk minimizer: $\widehat{F}_0(\cdot) = \arg\min_c \frac{1}{N}\sum_{i=1}^N L(Y_i, c)$, where c is a constant that minimizes the average loss.

2. For $m = 1$ to m_{stop},

   i Compute the negative gradient vector:
   $U_{i,m} = -\left[\frac{\partial L(Y_i, F(X_i))}{\partial F(X_i)}\right]_{F(X_i)=\widehat{F}_{m-1}(X_i)} \quad i = 1, \dots, N$

   ii Fit a real-valued base (weak) learner $\hat{f}_m$ to the gradient vector $U_{i,m}$.

   iii Find the step size (shrinkage) parameter ν_m as
   $\nu_m = \arg\min_\nu \sum_{i=1}^N L\left(Y_i, \widehat{F}_{m-1}(X_i) + \nu \hat{f}_m(X_i)\right)$

   iv Update the model: $\widehat{F}_m(\cdot) = \widehat{F}_{m-1}(\cdot) + \nu_m \hat{f}_m(\cdot)$

Finally, a crucial aspect of boosting algorithms is selecting an appropriate base procedure, often referred to as the base or weak learner, or simply the learner. This choice may be driven by the goal of improving predictive performance, while also considering the structural characteristics of the resulting boosting estimator. Indeed, the generic boosting estimator is formally expressed as

$$\widehat{F}_M(\cdot) = \widehat{F}_0(\cdot) + \sum_{m=1}^{M} \nu_m \hat{f}_m(\cdot), \tag{9}$$

that is, as a sum of base procedure estimates. In consequence, structural properties of the boosting function estimator are induced by a linear combination of the structural characteristics of the base procedure chosen (see Bühlmann and Van De Geer (2011)). Trees are the most commonly used base-learners in boosting, although linear models and splines are also frequently employed. For the study at hand, we will introduce GeD splines as the base procedure (see Section 5). But first, let us delve a bit deeper into the application of boosting.

4.1. Boosting for regression

For regression problems with continuous response $Y \in \mathbb{R}$, the squared-error loss—often referred to as L_2-loss—is the most frequently used loss function:

$$L\left(y, F(x)\right) = \frac{\left(y - F(x)\right)^2}{2}. \tag{10}$$

That is, the previously mentioned L_2Boost algorithm basically implements the general FGB algorithm presented in Algorithm 3 using (10) as loss function. The corresponding population minimizer of the L_2-loss is:

$$F^*(x) = \mathbb{E}[Y|X = x] \tag{11}$$

and thus L_2Boost leads to classical least squares regression of the mean. Note that, the squared-error loss aligns with the negative log-likelihood of the Gaussian distribution. Consequently, L_2Boosting is particularly suitable when the response follows a Normal distribution, since, in this case, it is tantamount to maximizing the likelihood of the data. In addition, L_2Boost benefits from a straightforward iterative structure given that the negative gradient of the squared-error loss is simply the residuals vector. As a result, the algorithm boils down to re-fitting the residuals from the previous boosting iteration at each new iteration.

If interested in the median of the conditional distribution instead, an alternative loss function is the L_1-loss, $L(y, F(x)) = |y - F(x)|$, with population minimizer $F^*(x) = \text{median}(Y|X = x)$. When dealing with limited sample sizes, squared-error loss heavily prioritizes observations with large absolute residuals, $|y_i - F(x_i)|$, during the modeling process. Hence, its effectiveness drastically reduces when faced with long-tailed error distributions or in the presence of "outliers". Absolute loss has been found to be more robust in such situations (see, e.g., Karunasingha (2022)).

Finally, if dealing with moderately heavy tails, as a compromise between the L_2 and L_1 loss, one may also consider the Huber-loss function. The latter provides strong resistance to gross outliers while being nearly as efficient as least squares for Gaussian errors (see Hastie *et al.* (2009)).

4.2. Boosting for classification

Boosting can be effectively applied in classification scenarios as well. Take the case where $Y \in \{0, 1\}$ is a binary response variable, and $Y|X = x \sim \text{Bernoulli}(p(x))$ with $p(x) = P(Y = 1|X = x)$. A common practice in binary classification is to model the posterior probability, $p(x)$, of an instance belonging to the positive class, through the log-odds (or logit) function:

$$F(x) = \text{logit}(p(x)) = \log\left(\frac{p(x)}{1 - p(x)}\right) = \beta_0 + \sum_{j=1}^{P} \beta_j x^{(j)},$$

which relates linearly to the predictor variables $X^1, \ldots, X^P$. Note that $F(x) > 0$ implies $p(x) > 1/2$, which is somewhat the "standard threshold" for binary classification in case of equal odds. In consequence, the natural classifier is simply

$$\mathcal{C}(x) = \begin{cases} 1 & \text{if } F(x) > 0 \\ 0 & \text{if } F(x) \leq 0 \end{cases}. \tag{12}$$

For ease of notation, it is often convenient to encode the response variable by $\tilde{Y} = 2Y - 1 \in \{-1, 1\}$. A misclassification therefore happens if $\tilde{Y}F(X) < 0$. The misclassification loss is thus given by

$$L(y, F(x)) = \mathbb{1}(\tilde{y}F(x) < 0), \tag{13}$$

whose corresponding population minimizer is equivalent to the Bayes classifier for $\tilde{Y}$:

$$F^*(x) = \begin{cases} +1 & \text{if } p(x) > 1/2 \\ -1 & \text{if } p(x) \leq 1/2 \end{cases}, \tag{14}$$

where the quantity $\tilde{y}F$ is the so-called "margin-value".

Nevertheless, note that the misclassification loss (13) cannot be used for FGB (Algorithm 3), since it is discontinuous and non-convex, both as a function of F as well as a function of the margin value $\tilde{y}F$. Therefore, gradient descent cannot be implemented neither in the space of function values nor in the space of margin values (see Friedman (2001)). Hence, for binary classification problems, it is most usual to use the negative binomial log-likelihood as loss function. This can be seen as a convex upper approximation of the misclassification loss (see Bühlmann and Van De Geer (2011)). Given a binary outcome variable $Y \in \{0,1\}$ with $Y|X = x \sim \text{Bernoulli}(p(x))$ and posterior probability $p(x) = P(Y = 1|X = x)$, the negative binomial log-likelihood of an instance is defined by:

$$L(y, p(x)) = -[y \log(p(x)) + (1 - y) \log(1 - p(x))].$$

Scaling, this is equivalent to

$$L(y, F(x)) = \log_2 (1 + \exp(-2\tilde{y}F(x))), \tag{15}$$

which is a convex and differentiable function in F. The corresponding population minimizer can be shown to be

$$F^*(x) = \frac{1}{2} \log \left(\frac{p(x)}{1 - p(x)} \right). \tag{16}$$

4.3. Boosting in the exponential family

Any negative log-likelihood function associated with an exponential family distribution can serve as a loss function in boosting (see Schmid and Hothorn (2008b)); consequently, FGB generalizes to the entire exponential family. For example, for count data with $Y \in \{0, 1, 2, \ldots\}$, we consider Poisson regression, that is, assume $Y|X = x \sim \text{Poisson}(\lambda(x))$, and implement FGB using as loss function:

$$L(y, F(x)) = -yF(x) + \exp(F(x)), \text{ where, } F^*(x) = \log(\lambda(x)). \tag{17}$$

Table 2 displays the loss function for some common distributions from the exponential family, along with their corresponding population loss/risk minimizer and negative gradient function.

4.4. Asymptotic properties of boosting

Bühlmann and Yu (2003) study the computationally simple L_2Boost algorithm and develop several theoretical findings related with the usage of cubic smoothing splines as base-learners. In particular, L_2Boost with smoothing splines achieves the minimax rate of convergence under the squared L_2 loss, for one-dimensional function estimation. As noted by Bühlmann and Hothorn (2007), this result is extended to much more general settings by Yao, Rosasco, and Caponnetto (2007) and Bissantz, Hohage, Munk, and Ruymgaart (2007). Bühlmann (2006) demonstrates the consistency of L_2Boosting for high-dimensional linear models where the number of predictors grows exponentially with the sample size, under the assumption of i.i.d. errors. This result relies on the ℓ_1-norm sparsity of the regression coefficients in the true underlying regression function. More recently, Yousuf and Ng (2021) extend this by establishing the consistency of L_2Boosting with componentwise local constant and linear estimators as base-learners, specifically for high-dimensional models with locally stationary predictors and polynomially decaying error tails. A brief review on the asymptotic theory developed for boosting is included in Section 9.2 of Bühlmann and Hothorn (2007), where further references can be traced. A more recent study on the asymptotic properties of boosting

Distribution of $Y \mid X = x$	Loss function $L(y, F(x))$	Risk minimizer $F^*(x)$	Negative gradient $U = -\partial L(y, F(x))/\partial F(x)$
Normal$(\mu(x), \sigma^2)$	$\frac{(y-F(x))^2}{2}$	$\mathbb{E}[Y \mid X = x]$	$y - F(x)$
Binomial$(1, p(x))$[a]	$\log_2(1 + \exp(-2\tilde{y}F(x))$	$\frac{1}{2}\log\left(\frac{p(x)}{1-p(x)}\right)$	$-\left(\frac{-2\tilde{y}\exp(-2\tilde{y}F(x))}{\log(2)\times(1+\exp(-2\tilde{y}F(x)))}\right)$
Gamma$(k(x), \theta)$[b]	$\log\Gamma(k) + kF(x) - k\log k$ $-(k-1)\log y + yke^{-F(x)}$	$\arg\min_{F(\cdot)} L(y, F(x))$	$k(1 - ye^{-F(x)})$
Poisson$(\lambda(x))$[c]	$-yF(x) + \exp(F(x))$	$\log(\lambda(x))$	$y - \exp(F(x))$

Table 2: Loss functions—with their respective population minimizer and negative gradient—for some commonly used distributions in FGB.

[a] With logit link function, $\log(\frac{\mu}{1-\mu}) = \eta$.

[b] Negative Gamma log-likelihood with logarithmic link function, $\log(\mu) = \eta$. Response should be non-negative and continuous. Note there is no analytical solution for the risk minimizer. The parameter k is initialized to $k = 1$. Then, before recomputing the negative gradient at each boosting iteration, it is re-estimated as $k = \arg\min_k L(y, \hat{F}_{m-1}(x))$.

[c] Negative Poisson log-likelihood with the natural link function, $\log(\mu) = \eta$.

is provided by Liang and Sur (2022), where a comprehensive literature review can be found in Sections 1 and 4.

5. L_2Boost Normal GeD spline regression

As discussed in the previous section, boosting algorithms require the specification of a base procedure (base/weak learner) to fit the gradient vector at each boosting iteration. In this respect, GeDS constitutes a promising alternative. Gradient boosting, as previously mentioned, is a machine learning ensemble technique designed to enhance predictive accuracy by aggregating the predictions of multiple weak learners. Typically, the final boosted model is expressed as the cumulative sum of all the sub-models generated across successive boosting iterations (cf. (9)). This approach, however, entails two major drawbacks: the computational burden of summing a large number of base-learner fits when evaluating the model and, more critically, the loss of model interpretability. In contrast, utilizing GeD regression splines as base-learners allows the (partial) sums of these learners to be condensed into a single, explicit spline regression model (see Step 3 in Algorithm 4). This represents a significant advantage of incorporating GeDS into FGB, as it simplifies the evaluation of the final boosted model and enhances its interpretability. Specifically, we leverage the ability of re-expressing the B-spline representation of a GeD base-learner into piecewise polynomial form. This allows for the direct summation of polynomial coefficients of the corresponding segments within the boosting algorithm. Note that such a transformation is not feasible when using, e.g., smoothing splines as base-learners (cf. Schmid and Hothorn (2008a)), where the number of boosting iterations required is significantly higher (see Sections 6 and 8).

5.1. L_2-GeDS Boost algorithm

For simplicity, we introduce FGB-GeDS through the L_2-GeDS Boost algorithm for fitting a non-additive (i.e., single base-learner) model. This implements functional gradient boosting employing the squared-error loss as loss function and a single GeDS base-learner. At each boosting iteration, the negative gradient vector—which in this context corresponds to the residuals vector—is fitted using the base procedure encapsulated within the `NGeDS()` function from the **GeDS** package, which constructs a GeD spline regression model for a Normal response variable.

Algorithm 4 L_2-GeDS Boost

Step 1.

1.a. Given data $\{Y_i, X_i\}_{i=1}^N$, fit a GeDS linear model, $Y \sim \hat{f}_0(X)$[1], with κ_0 internal knots (initial learner):

$$\widehat{F}_0\left(\boldsymbol{\Delta}_{d_0,2};\cdot\right) = \hat{f}_0\left(\boldsymbol{\delta}_{\kappa_0,2}, \hat{\boldsymbol{\alpha}}_p;\cdot\right)$$

where $\boldsymbol{\delta}_{\kappa_0,2}$ denotes the set of knots of the linear base-learner *GeDS fit* (i.e. of order $n = 2$) and $\hat{\boldsymbol{\alpha}}_p$ is the respective vector of $p = \kappa_0 + 2$ estimated B-spline coefficients. We will denote by $\boldsymbol{\Delta}_{d_m,2} = \{\xi_1 = \xi_2 < \xi_3 < \ldots < \xi_{d_m+3} = \xi_{d_m+4}\}$ the set of knots of the linear L_2*-GeDS Boost fit* at each boosting iteration, m; this consists of the ordered distinct knots from the pooled set $\{\boldsymbol{\delta}_{\kappa_0,2} \cup \ldots \cup \boldsymbol{\delta}_{\kappa_m,2}\}$.

Convert the linear GeDS fit—which has a B-spline representation—into piecewise polynomial form, defined by the distinct knots $\{\delta_2 < \delta_3 < \ldots < \delta_{\kappa_0+2} < \delta_{\kappa_0+3}\} \in \boldsymbol{\delta}_{\kappa_0,2}$ and the corresponding polynomial coefficients. Given κ_0 internal knots there are $\kappa_0 + 1$ polynomial pieces/intervals; consequently, this initial linear GeDS fit is characterized by $\kappa_0 + 1$ pairs of polynomial coefficients (i.e., intercept and slope) corresponding to each of these intervals. Set $m = 0$ and initialize the coefficients of the linear boosted GeDS model as

$$(a_m, b_m) = (a_m^\dagger, b_m^\dagger)$$

where $(a_m^\dagger, b_m^\dagger)$ denote, respectively, the intercept and slope vectors of coefficients from the piecewise polynomial representation of the linear *GeDS fit*, $\hat{f}_0\left(\boldsymbol{\delta}_{\kappa_0,2}, \hat{\boldsymbol{\alpha}}_p;\cdot\right)$—i.e., the base-learner fit in this first step. Conversely, we use (a_m, b_m) to denote the intercept and slope vectors of coefficients from the piecewise polynomial representation of the linear L_2*-GeDS Boost fit*, $\widehat{F}_m\left(\boldsymbol{\Delta}_{d_0,2};\cdot\right)$, which are updated at each boosting iteration.

1.b. Compute the negative gradient vector of the loss function and evaluate it at $\widehat{F}_m\left(\boldsymbol{\Delta}_{d_m,2}; X_i\right)$. For the L_2 loss, this corresponds to the residuals vector:

$$U_{i,m+1} = -\left[\frac{\partial L(Y_i, F(X_i))}{\partial F(X_i)}\right]_{F(X_i)=\widehat{F}_m(\cdot)} = Y_i - \widehat{F}_m\left(\boldsymbol{\Delta}_{d_m,2}; X_i\right) = r_{i,m+1}, \tag{18}$$

with $i = 1, \ldots, N$.

[1]For simplicity, we present the algorithm in terms of univariate surface smoothers (i.e., $X \in \mathbb{R}^{N\times 1}$), although `NGeDS()`/`GGeDS()` also support bivariate spline regression (i.e., X may also lie in $\mathbb{R}^{N\times 2}$).

Step 2. Increase m by one: $m \leftarrow m+1$. Fit a linear GeDS model with κ_m internal knots, $\boldsymbol{\delta}_{\kappa_m,2}$, to the residuals, $\{r_{i,m}\}_{i=1}^N$. Convert the fitted GeDS learner from its B-spline representation to piecewise polynomial form. Extract the breakpoints, $\{\delta_2 < ... < \delta_{\kappa_m+3}\}$, from $\boldsymbol{\delta}_{\kappa_m,2}$, and identify the corresponding polynomial coefficients. We denote the intervals defined by these breakpoints as $\mathcal{I}^{GeDS} = \left\{\mathcal{I}_j^{GeDS} = [\delta_{j+1}, \delta_{j+2}),\ j = 1,..,\kappa_m + 1\right\}$.

Step 3. Let $\mathcal{I}^{m-1} = \left\{\mathcal{I}_i^{m-1} = [\xi_{i+1}, \xi_{i+2}),\ i = 1,..,d_{m-1} + 1\right\}$, $d_{m-1} + 1 \leq \sum_{l=0}^{m-1}(\kappa_l + 1)$, be the collection of intervals corresponding to the piecewise representation of the linear L_2-GeDS Boost fit at the $(m-1)$th iteration. Recompute each of the pairs of coefficients $(a_m^{(k)}, b_m^{(k)})$ of the boosted model as,

$$\begin{cases} a_m^{(k)} = a_{m-1}^{(i)} + \nu \times a^{\dagger}{}_m^{(j)} \\ b_m^{(k)} = b_{m-1}^{(i)} + \nu \times b^{\dagger}{}_m^{(j)} \end{cases} \quad \text{if } \mathcal{I}_i^{m-1} \cap \mathcal{I}_j^{GeDS} \neq \emptyset$$

where $0 < \nu \leq 1$ is a real valued step length/shrinkage factor[2]. Note that, at each iteration, $(a_m^{\dagger}, b_m^{\dagger})$ will have dimensions $(\kappa_m+1)\times 2$, while (a_m, b_m) will have dimensions $(d_m + 1) \times 2$. So, in other words, update the model as:

$$\widehat{F}_m\left(\boldsymbol{\Delta}_{d_m,2}; \cdot\right) = \widehat{F}_{m-1}\left(\boldsymbol{\Delta}_{d_{m-1},2}; \cdot\right) + \nu \times \hat{f}_m\left(\boldsymbol{\delta}_{\kappa_m,2}, \hat{\boldsymbol{\alpha}}_p; \cdot\right)$$

where $\boldsymbol{\Delta}_{d_m,2}$ is obtained from the ordered distinct knots in the pooled set of knots $\left\{\boldsymbol{\Delta}_{d_{m-1},2} \cup \boldsymbol{\delta}_{\kappa_m,2}\right\}$. See Appendix A for a detailed algorithm.

Step 4. Recompute the residuals, as specified in (18), with respect to the model $\widehat{F}_m\left(\boldsymbol{\Delta}_{d_m,2}; \cdot\right)$. Let $q_{boost} \geq 1$ be some predefined integer. If $m < q_{boost}$ go back to step 2, otherwise calculate the ratio:

$$\phi_{boost} = \text{RSS}(m)/\text{RSS}(m - q_{boost})$$

with $\text{RSS}(m) = \sum_{i=1}^N r_{i,m}^2$ being the residual sum of squares.

Step 5. Repeat iteratively steps $2-4$ until $\phi_{boost} \geq \phi_{boost}^{exit}$, where $\phi_{boost}^{exit} \in (0,1)$ is some threshold level chosen to be close to 1. Note that it is not guaranteed that $\text{RSS}(m) < \text{RSS}(m-1)$, hence ϕ_{boost} could be greater than 1. The ratio ϕ_{boost} will be close to 1 if no (or very little) improvement has been achieved in the last q_{boost} consecutive boosting iterations, meaning that the corresponding values of the RSS have stabilized. And it will be greater than 1 if the model is worsening as the boosting iterations continue.

Step 6. Since ϕ_{boost} may be greater than 1 we set the final model to be the one that, among the last q_{boost} models, minimizes the empirical residual sum of squares:

$$\widehat{F}^*\left(\boldsymbol{\Delta}_{d_{m^*},2}; \cdot\right) = \underset{\widehat{F}_l(\boldsymbol{\Delta}_{d_l,2};\cdot)}{\arg\min} \sum_{i=1}^N \left(Y_i - \widehat{F}_l(\boldsymbol{\Delta}_{d_l,2}; X_i)\right)^2, \quad l = m - q_{boost}, ..., m.$$

[2]As suggested by Bühlmann and Van De Geer (2011), we assume a constant ν.

where m^* represents the boosting iteration corresponding to the final linear L_2-GeDS Boost fit.

Step 7. Consider the final linear fit $\widehat{F}^*\left(\boldsymbol{\Delta}_{d_{m^*},2};\cdot\right)$ obtained in step 6. This has d_{m^*} internal knots, with locations given by $\boldsymbol{\Delta}_{d_{m^*},2}$. We compute the higher-order fits, specifically for $n = 3$ and $n = 4$, by first calculating the knot placement $\bar{\boldsymbol{\tau}}_{*,n}$[3]. This is defined as:

$$\tau_{i+n} = \left(\Delta_{i+2} + \ldots + \Delta_{i+n}\right)\Big/(n-1), \quad i = 1, \ldots, d_{m^*} - (n-2) \tag{19}$$

And second, we find the least squares fit $\widehat{F}^*\left(\bar{\boldsymbol{\tau}}_{*,n}, \hat{\boldsymbol{\theta}};\cdot\right)$ that solves

$$\min_{\boldsymbol{\theta}} \sum_{1\le i\le N} \left(Y_i - F^*\left(\bar{\boldsymbol{\tau}}_{*,n}, \boldsymbol{\theta}; X_i\right)\right)^2.$$

Hence, in the same spirit as the canonical GeDS method, in addition to the final linear fit, final quadratic and cubic fits are also obtained.

If $\phi_{boost} \ge \phi_{boost}^{exit}$, the performance of $\widehat{F}_m\left(\boldsymbol{\Delta}_{d_m,2};\cdot\right)$ does not significantly improve (and may even worsen) with q_{boost} additional iterations. Consequently, the iterations are stopped, and $\widehat{F}^*\left(\boldsymbol{\Delta}_{d_{m^*},2};\cdot\right)$ is selected as the linear model that adequately captures the "shape" of the unknown underlying function F. Quadratic and cubic fits are then computed using the boosted knot vector $\boldsymbol{\Delta}_{d_m,2}$, correspondingly transformed by the so-called *averaging knot location* method described in Kaishev *et al.* (2016) and formalized in (19). Maximum likelihood coefficients for these fits are subsequently estimated. Note that, in the linear FGB-GeDS fit, both knots and coefficients are estimated during the boosting process. In contrast, for higher-order fits, the FGB iterations serve exclusively to select the knot vector.

The FGB-GeDS method offers flexible control over the strength of the base-learners. In particular, the suggested approach—implemented in the function `NGeDSboost()`, presented in Section 5.2—is as follows: start with a weak GeDS initial learner (i.e., up to 2 maximum internal knots) and then perform a few boosting iterations using GeDS learners operating at their full potential (i.e., without fixing a maximum number of knots); that is, pre-define the maximum number of internal knots for the initial learner, $\kappa_0^{\max}$, and then, at each boosting iteration, allow for the strength of the base-learner—i.e., the number of knots κ_m of the learner $\hat{f}_m\left(\boldsymbol{\delta}_{\kappa_m,2}, \hat{\boldsymbol{\alpha}}_p;\cdot\right)$—to be automatically regulated by the GeDS method. This can be tuned using the GeDS parameters (ϕ, β, q) presented in Section 2, and discussed in further detail in Kaishev *et al.* (2016).

Note that, on the one hand, the optimal number of boosting iterations is automatically determined by a stopping rule, consisting of a ratio of deviances of consecutive models (c.f. Step 4 in Algorithm 4. On the other hand, the strength of the GeDS base-learner at each boosting iteration is automatically regulated by the GeDS technique itself, and thus it is usually not necessary to use the shrinkage parameter ν to regulate the strength of the base-learner at each boosting iteration (i.e., $\nu = 1$), though in certain cases it may provide additional flexibility (see Examples 8.1 and 8.3).

[3]The choice of the knots $\bar{\boldsymbol{\tau}}_{*,n}$, according to Kaishev *et al.* (2016), ensures that the nth order spline predictor curve $\widehat{F}^*\left(\bar{\boldsymbol{\tau}}_{*,n};\cdot\right)$ becomes nearly the VDS approximation to $\widehat{F}^*\left(\boldsymbol{\Delta}_{d_{m^*},2};\cdot\right)$.

5.2. Component-wise L_2-GeDS boosting

Boosting algorithms can also be regarded as stagewise techniques for fitting additive models (see Friedman *et al.* (2000)). In particular, Bühlmann and Yu (2003) introduce component-wise (or model-based) boosting, a variant of gradient boosting to construct additive models by selectively incorporating predefined base-learners, each involving one or multiple features. At each iteration, the model is updated in a component-wise fashion: each single base-learner of the additive model is independently fitted and the algorithm identifies and updates solely the base-learner whose update reduces the loss function the most (see Potts, Bergherr, Reinke, and Griesbach (2023)). This method inherently performs variable selection by incorporating one base-learner at a time into the ensemble, allowing the same or different learners to be added across iterations.

Let us formally present component-wise gradient boosting as follows. Consider the random sample of i.i.d random variables, $\{Y_i, X_i\}_{i=1}^N$, where $Y_1, ..., Y_N$ are one-dimensional response vectors and $X_1, ..., X_N$ are P-dimensional vectors of covariates. Given this data sample and a pre-chosen set of K univariate or multivariate base-learners (i.e. $K \leq P$), the objective is to estimate F^* as:

$$\hat{F}^* = \hat{F}_0 + \hat{F}_1^* + ... + \hat{F}_K^* \quad \text{with} \quad \hat{F}_j^* = \nu \times \sum_{i=1}^{m^*} f_i^j \mathbb{1}\left(\hat{\boldsymbol{f}}_{\boldsymbol{i}} = \hat{f}_i^j\right), \quad j = 1, ..., K \tag{20}$$

where m^* stands for the boosting iteration where the optimal final fit $\hat{F}^*$ is achieved; $\hat{\boldsymbol{f}}_{\boldsymbol{i}}$ denotes the optimal learner selected at the ith boosting iteration, according to some pre-established criterion. Each function estimate $\hat{F}_j$ is then the cumulative sum of the estimates f_i^j for each iteration where the corresponding base-learner, j, was selected to update the model $\widehat{F}$, i.e., for each iteration where $\hat{\boldsymbol{f}}_{\boldsymbol{i}} = \hat{f}_i^j$. Algorithm 5 describes our implementation of component-wise gradient boosting using the L_2 loss function and GeD splines as base-learners.

Algorithm 5 Component-wise L_2-GeDS Boost

Step 1.

1.a. Consider the sample $\{Y_i, X_i\}_{i=1}^N$ where each Y_i is a one-dimensional response and each X_i is a vector of P features. Consider also a collection of K base-learners (this can be either univariate or bivariate). For simplicity, let us consider the case where there is exactly one (GeDS) base-learner per predictor variable, i.e., $K = P$. Fit a linear GeDS model $Y \sim \hat{f}_0^j(X^j)$ with κ_0^j internal knots for each of the K components, $j = 1, ..., K$. Set $\widehat{F}_0$ to be the base-learner fit that minimizes the sum of squared residuals, that is:

$$\widehat{F}_0\left(\boldsymbol{\Delta}_{d_0,2};\cdot\right) = \underset{\substack{\hat{f}_0^j \\ j=1,...,K}}{\arg\min} \sum_{i=1}^N \left(Y_i - \hat{f}_0^j(\boldsymbol{\delta}_{\kappa_0^j,2}; \hat{\boldsymbol{\alpha}}_{p^j}; X_i^j)\right)^2$$

Initialize $m = 0$.

1.b. Compute the negative gradient vector of the L_2 loss function (i.e. the residuals):

$$r_{i,m+1} = Y_i - \widehat{F}_m(X_i), \quad i = 1, ..., N$$

Step 2. Increase m by one: $m \leftarrow m+1$. Fit each of the K base-learners (linear GeDS models) to the residuals vector. Select the fit that minimizes the RSS:

$$\hat{\boldsymbol{f}}_{\boldsymbol{m}} = \underset{\hat{f}_m^j(\cdot)}{\arg\min} \sum_{i=1}^{N} (r_{i,m} - \hat{f}_m^j(\boldsymbol{\delta}_{\kappa_m^j,2}; \hat{\boldsymbol{\alpha}}_{p^j}; X_i^j))^2$$

Step 3. Update the current estimate as

$$\widehat{F}_m\left(\boldsymbol{\Delta}_{d_m,2};\cdot\right) = \widehat{F}_{m-1}\left(\boldsymbol{\Delta}_{d_{m-1},2};\cdot\right) + \nu \times \hat{\boldsymbol{f}}_{\boldsymbol{m}}$$

where $0 < \nu \leq 1$ is a real valued step length/shrinkage factor. The fit of the base-learner that "best" fitted the residuals vector should be updated in the piecewise manner discussed in step 3 of Algorithm 4.

Step 4. Recompute the residuals. Let $q_{boost} \geq 1$ be some prefixed integer. If $m < q_{boost}$ go back to step 2, otherwise calculate the ratio:

$$\phi_{boost} = \text{RSS}(m)/\text{RSS}(m - q_{boost})$$

Step 5. Repeat iteratively steps 2-5 until $\phi_{boost} \geq \phi_{boost}^{exit}$, where $\phi_{boost}^{exit} \in (0, 1)$, and is chosen to be close to 1.

Step 6. Since ϕ_{boost} may be greater than 1 we set the final model to be the one that minimizes the empirical residual sum of squares:

$$\widehat{F}^*\left(\boldsymbol{\Delta}_{d_{m^*},2};\cdot\right) = \underset{\widehat{F}_l(\boldsymbol{\Delta}_{d_l,2};\cdot)}{\arg\min} \sum_{i=1}^{N} \left(Y_i - \widehat{F}_l(\boldsymbol{\Delta}_{d_l,2}; X_i)\right)^2, \quad l = m - q_{boost}, ..., m.$$

Step 7. Step 6 yields a final linear fit, $\widehat{F}^*\left(\boldsymbol{\Delta}_{d_{m^*},2};\cdot\right)$, that, for each of the K base-learners, has $d_{m^*}^j$, $d_{m^*}^j \leq \sum_{i=0}^{m_j^*} \kappa_i^j$, internal knots with knots locations $\boldsymbol{\Delta}_{d_{m^*},2}^j$, where $m_j^* = \sum_{m=0}^{m^*} \mathbb{1}_{\hat{\boldsymbol{f}}_{\boldsymbol{m}} = \hat{f}_m^j}$. We compute the higher order fits (quadratic and cubic) calculating for each base-learner the knot placement $\bar{\boldsymbol{\tau}}_{*,n}^j$ defined as:

$$\tau_{i+n}^j = (\Delta_{i+2}^j + ... + \Delta_{i+n}^j)\Big/(n-1), \quad i = 1, ..., d_{m^*}^j - (n-2).$$

and finding the least squares fit $\widehat{F}^*\left(\bar{\boldsymbol{\tau}}_{*,n}, \hat{\boldsymbol{\theta}};\cdot\right)$ that solves

$$\min_{\boldsymbol{\theta}} \sum_{1 \leq i \leq N} \left(Y_i - F^*\left(\bar{\boldsymbol{\tau}}_{*,n}, \boldsymbol{\theta}; X_i\right)\right)^2 \quad \text{where} \quad \bar{\boldsymbol{\tau}}_{*,n} = \cup_{j=1}^{K} \bar{\boldsymbol{\tau}}_{*,n}^j.$$

The function NGeDSboost() in the **GeDS** package implements functional gradient boosting with GeD splines, following Algorithm 4, when the model provided has a single base-learner. If an additive model is provided instead, it implements component-wise functional gradient boosting with GeD splines, following Algorithm 5. The synopsis of this function is the following:

```
NGeDSboost(formula, data, weights = NULL, normalize_data = FALSE,
family = mboost::Gaussian(), link = NULL,
initial_learner = TRUE, int.knots_init = 2L,
min_iterations, max_iterations, shrinkage = 1,
phi_boost_exit = 0.99, q_boost = 2L,
beta = 0.5, phi = 0.99, int.knots_boost = 500L, q = 2L,
higher_order = TRUE, boosting_with_memory = FALSE)
```

The model is specified using a formula of the type `Y~ f(X_1)+f(X_2)+...` and the data sample to be fitted should be provided as `data.frame` via the `data` argument. Data can be standardized before fitting by setting `normalize_data = TRUE`, and a maximum and minimum number of boosting iterations can be set through `max_iterations` and `min_iterations`. `phi_boost_exit` and `q_boost` are the tuning parameters of the boosting iterations stopping rule (Step 4 in Algorithm 4/5). The base-learner(s) can be tuned by the GeDS parameters `beta`, `phi` and `q`. The maximum number of internal knots of the initial learner κ_0^{max} is set via `int.knots_init`, and the maximum number of internal knots of the base-learner(s) at each boosting iteration κ_m^{max} can be set via `int.knots_boost`.

6. Numerical examples – non-additive models

In this section, we introduce the use of the **GeDS** package for fitting non-additive univariate spline models. For this purpose, we consider the canonical and boosted GeDS methods, which have been presented in Sections 2 and 5, respectively. These methods are implemented by the NGeDS()/GGeDS() and NGeDSboost() functions. We begin with a simulated data example that we will use to illustrate the FGB-GeDS fitting process detailed in Algorithm 4. Following this, we provide some additional examples to conduct a comparative analysis between the GeDS models and the boosting with P-splines implementation from the **mboost** package.

6.1. FGB-GeDS fitting process

Example 6.1 We assume the "true" linear predictor to be $\eta = f_1(x)$, where,

$$f_1(x) = 40\frac{x}{1+100x^2} + 4, \quad x \in [-2, 2] \tag{21}$$

we then generate random samples, $\{X_i, Y_i\}_{i=1}^N$ with corresponding Normal, Poisson and Gamma distributed response variable, Y, and uniformly distributed explanatory variable, X, i.e., $Y_i \sim N(\mu_i, \sigma)$ with $\sigma = 0.2$, $\mu_i = \eta_i = f_1(X_i)$; $Y_i \sim \text{Poisson}(\mu_i)$ with $\mu_i = \exp\{\eta_i\}$ and $\eta_i = f_1(X_i)$; $Y_i \sim \text{Gamma}(\mu_i, \varphi)$ with $\varphi = 0.1$, $\mu_i = \exp\{\eta_i\}$ and $\eta_i = f_1(X_i)$; and $X_i \sim U[-2, 2]$, $i = 1, ..., N$, where N is the sample size. In particular, we set $N = 500$. The R implementation of this example is as follows:

```
R> # Example 6.1
R> # Generate a data sample for the response variable Y
R> # and the covariate X
R> set.seed(123)
R> N <- 500
R> f_1 <- function(x) (10*x/(1+100*x^2))*4+4
R> X <- sort(runif(N, min = -2, max = 2))

R> # Normal
R> means <- f_1(X)
R> # Add (Normal) noise to the mean of Y
R> Y <- rnorm(N, means, sd = 0.2)

R> # Poisson and Gamma
R> means <- exp(f_1(X))
R> # Generate Poisson distributed Y according to the mean model
R> Y <- rpois(N, means)
R> # Generate Gamma distributed Y according to the mean model
R> Y <- rgamma(N, shape = means, rate = 0.1)
```

Figure 1 illustrates the fitting stages for the Normal version of example 6.1 of an FGB-GeDS model with $\kappa_0^{\max} = 2$, that is, with an `NGeDS()` fit using (at most) two internal knots as the initial learner. In the left column, the linear fit of the model to the data at each boosting iteration is depicted, starting with the initial `NGeDS()` fit with (at most) two internal knots. Above each plot, the corresponding vector of internal knots, $\boldsymbol{\Delta}_{d_m,2}$, of each fit is displayed. On the right, the `NGeDS()` base-learner fit to the recomputed residuals at each boosting iteration is presented; the internal knots fitted at the corresponding iteration are displayed at the top of each plot. The final plot displays the linear fit, alongside the quadratic and cubic fits. The quadratic and cubic fits are obtained by first relocating the knots, using Equation (19), to ensure the predictor curve becomes nearly the VDS approximation to the linear fit, and then re-estimating the B-spline coefficients via least squares, as described in Step 7 of Algorithm 4.

Figure 1 is obtained by running the following code snippet:

```
R > Gmodboost <- NGeDSboost(Y ~ f(X), data = data)
R> par(mfrow=c(4,2))
R> visualize_boosting(Gmodboost, 0:Gmodboost$iters, final_fits = TRUE)
```

6.2. Further examples and model comparison

Bühlmann and Yu (2003) empirically demonstrate the competitiveness of L_2Boosting with componentwise smoothing splines compared to conventional estimation methods, like back-fitting or boosting with trees. Schmid and Hothorn (2008a) replace smoothing spline base-learners by P-spline base-learners, which yield approximately the same performance results but are more advantageous from a computational perspective.

The choice of two main boosting parameters is discussed in Bühlmann and Hothorn (2007), Schmid and Hothorn (2008a) and Bühlmann and Van De Geer (2011), namely, the stopping

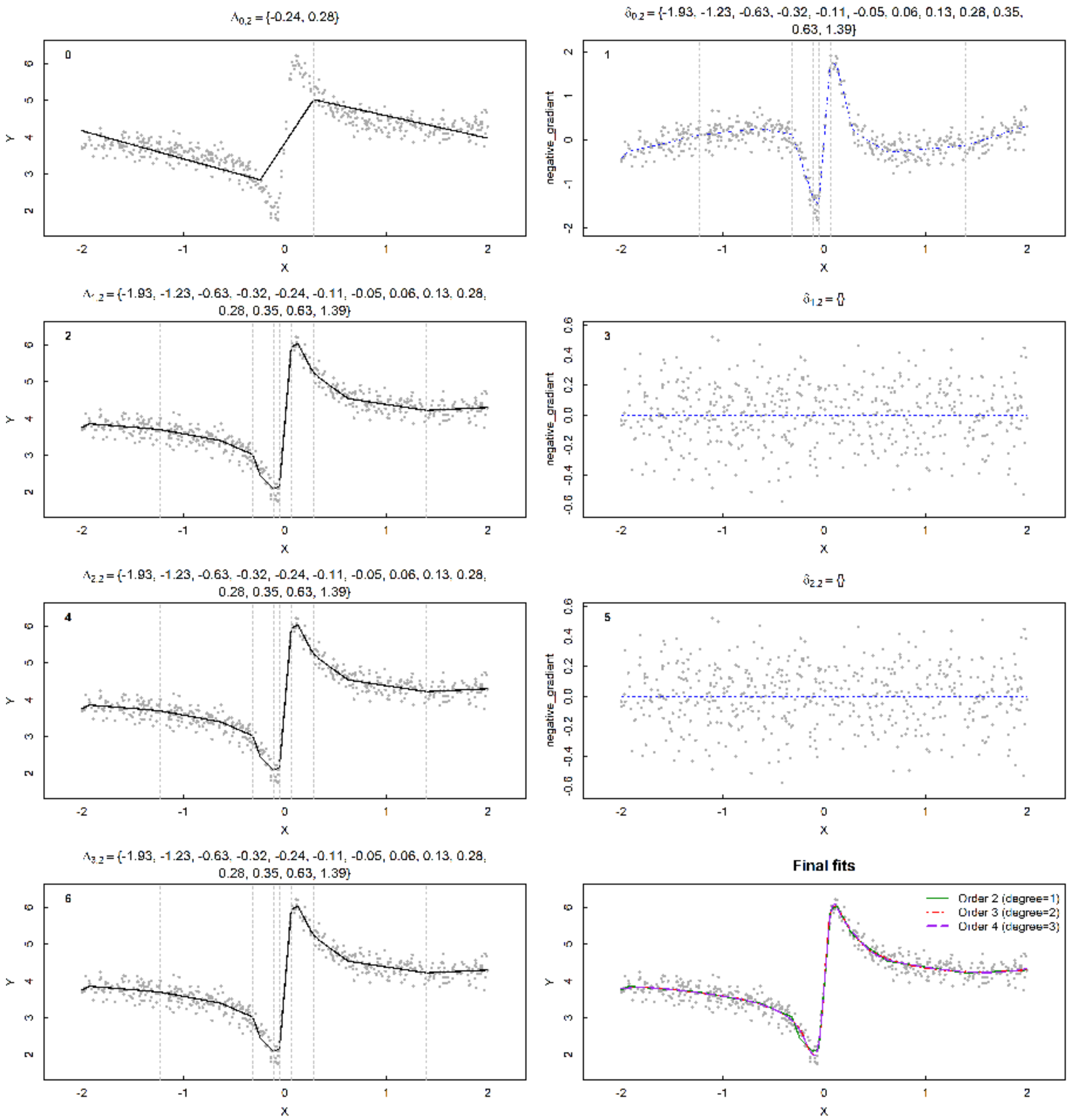


Figure 1: *(Left)* Linear `NGeDSboost()` fit over the data at each boosting iteration. *(Right)* `NGeDS()` fit (i.e. base-learner fit) to the recomputed residuals at each boosting iteration. The last plot depicts the final linear fit together with the higher order fits (quadratic and cubic).

boosting iteration, m_{stop}, and the step length factor (shrinkage rate), ν. On the one hand, selecting an appropriate m_{stop} value is crucial to prevent data overfitting. These authors recommend using an early-stopping strategy to maximize prediction accuracy. The latter may rely on cross-validation techniques or, less preferably, AIC-based methods, since these tend to overshoot the optimal m_{stop} (see Hastie (2007), Hofner, Mayr, Robinzonov, and Schmid (2014), or Mayr *et al.* (2012)). On the other hand, the choice of the step length/shrinkage parameter ν is considered to be relatively less critical for the predictive performance of a boosting algorithm, provided it is set to a small value (e.g., $\nu = 0.01$ or $\nu = 0.1$; see Bühlmann and Hothorn (2007)).

However, as Schmid and Hothorn (2008b) suggest, cross-validation can be computationally expensive and time-consuming, especially with large datasets and complex models. In addition, particularly in the case of small datasets, the results from cross-validation can significantly vary depending on the number of folds and the method of partitioning the data, potentially leading to inconsistent determination of the optimal m_{stop}. Overall, determining m_{stop} via cross-validation inherently constitutes an *ex-post* approach, and thus may lead to analytical incongruences. In light of these challenges, the stopping rule presented in Step 4 of Algorithm 4 emerges as a robust and coherent method, effectively circumventing the latter drawbacks. Additionally, under the default FGB-GeDS approach presented in Section 5.1, the strength/weakness of the base-learner(s) at each boosting iteration is automatically regulated by the GeDS technique itself. Therefore, adjusting the step length/shrinkage parameter ν is not necessary in most occasions.

As follows we compare the performance of FGB-GeDS with the canonical GeDS method and with the boosting with P-splines procedure proposed by Schmid and Hothorn (2008a) and implemented in the R package **mboost**[4]. First, we consider two simulated examples: 6.1, which we have just introduced, and 6.2, introduced as follows. Second, we consider a real data application in 6.3.

Example 6.2 - Doppler function. We assume that the "true" linear predictor, $\eta = f_2(x)$, where

$$f_2(x) = 5\sqrt{x(1-x)}\sin\frac{2\pi(1+0.05)}{(x+0.05)}, \quad x \in [0,1] \tag{22}$$

is the well-known Doppler function used by a number of authors such as Kaishev *et al.* (2016), Yang and Hong (2017) or Dimitrova *et al.* (2023). We simulate random samples, $\{X_i, Y_i\}_{i=1}^N$, with $Y_i \sim N(\mu_i = f_2(X_i), \sigma = 0.2)$ and $X_i = (i-1)/N$, $i = 1, ..., N$, $N = 400$.

```
R> # Example 6.2 - Doppler function
R> # Generate a data sample for the response variable
R> # Y and the single covariate X
R> set.seed(123)
R> N <- 400
R> f_2 <- function(x) {
+   5 * sqrt(x * (1-x)) * sin(2 * pi * (1 + 0.05) / (x + 0.05) )
+ }
R> X <- (1:N - 1) / N
R> # Specify a model for the mean of Y to include only a component
R> # non-linear in X, defined by the function f_2
R> means <- f_2(X)
R> # Add (Normal) noise to the mean of Y
R> Y <- rnorm(N, means, sd = 0.2)
```

We now compare the performance of `NGeDS()`/`GGeDS()`, `NGeDSboost()` and `mboost()` for examples 6.1 and 6.2. Before proceeding, let us review the parameters used in each model.

[4]Note that `bns()` (penalized natural splines) and `bss()` (smoothing splines) are deprecated (and no longer available) in **mboost**. Instead, **mboost** suggests using `bbs()` (P-spline base-learners), which results in "qualitatively the same models but is computationally much more attractive".

The parameters of `NGeDS()`/`GGeDS()` and `NGeDSboost()` for these two examples were chosen as follows. The value of β (one of the tuning parameters of stage A of GeDS) is set to 0.5 in example 6.1, and to 0.6 in example 6.2. These are within the range of recommended values for β in the case of "wiggly underlying functions and high signal-to-noise ratio data" by Kaishev *et al.* (2016). In particular, $\beta = 0.5$ means that, when determining the location of a new knot, the within-cluster mean residual value and the cluster range are considered equally important, while $\beta = 0.6$ implies that a slightly higher weight is put on the within-cluster mean residual value (c.f. Kaishev *et al.* (2016)). The tuning parameters for the stopping rule in stage A of GeDS, ϕ_{exit} and q, are set to their default values of 0.99 and 2, respectively. Regarding the specific parameters of `NGeDSboost()`, these are set to default, except for the shrinkage rate, ν, which is modified for the Poisson and Gamma examples. Therefore, $\phi_{boost}^{exit} = 0.99$ and $q_{boost} = 2$, for both examples; also, `initial_learner = TRUE` and `int.knots_init = 2`, which means that the FGB algorithm is run utilizing a GeDS initial learner with a maximum of 2 internal knots. In the Poisson and Gamma versions of Example 6.1, the shrinkage rates are set to $\nu = 0.005$ and $\nu = 0.1$, respectively. These values are chosen to account for the sensitivity of the logarithmic link function associated with these distributions. The lower shrinkage rate prevents overly large updates resulting from the exponential scale of the response for these distributions, hence ensuring stable updates at each boosting iteration. For the Normal version of example 6.1 and for example 6.2, the shrinkage is left at its default value, i.e., $\nu = 1$.

The function `mboost()` is run utilizing a cubic smooth P-spline as the base-learner. Two alternatives for this model are considered. First, we run an `mboost()` model in which both the number of knots and the degrees of freedom (d.f.) of the base learner are pre-tuned to minimize the median mean squared error (MSE) relative to the corresponding true functions, based on the fits to 10 simulated data samples. Second, for the sake of comparability, we also consider an alternative scenario in which only the d.f. are tuned, while the number of knots in the P-spline is fixed to match the median number across the simulations used by `NGeDSboost()` in each example. Following the recommendations in the **mboost** package documentation, we set the shrinkage rate to 0.1. For the Poisson example, however, we adjusted it to 0.05, as we did in `NGeDSboost()`, which improves performance compared to the default rate of 0.1. We also increased m_{stop} to $10,000$ boosting iterations. This adjustment resulted in better outcomes for these examples than the default setting of $m_{stop} = 100$.

The models are tested for examples 6.1 and 6.2, by fitting $1,000$ different simulated data sets. The R structure of the models compared is the following:

```
R> NGeDS(Y ~ f(X), beta)
R> GGeDS(Y ~ f(X), family, beta)

R> NGeDSboost(Y ~ f(X), data, family, shrinkage, beta)

R> mboost(Y ~ bbs(X, knots, degree = 3, df), data, family,
+   control = boost_control(mstop = 10000, nu))
```

In the normal version of 6.1 and in example 6.2, fits are compared according to their MSE with respect to the true generating function, defined as $\left\{\sum_{i=1}^{N}\left(f(X_i) - \hat{f}(X_i)\right)^2\right\}\Big/\mathrm{N}$. Similarly, for the Poisson and Gamma versions of 6.1, we assess the fits using the corresponding Poisson and Gamma mean deviance relative to the true generating function.

	Type	MSE/Mean Dev.	Boosting iter.	Internal knots	Time (sec.)
Example 6.1					
Normal					
`NGeDS()`	Linear	0.002710		10	0.06
	Quadratic	0.001733			
	Cubic	0.001690			
`NGeDSboost()`	Linear	0.002734	3	12	0.11
	Quadratic	0.001477			
	Cubic	0.001353			
`mboost()`	Tuned d.f & knots	0.003379	10,000	42	2.53
	Tuned d.f	0.051910	10,000	12	1.54
Poisson					
`NGeDS()`	Linear	0.094117		14	0.13
	Quadratic	0.052941			
	Cubic	0.049332			
`NGeDSboost()`	Linear	2.601673	4	17	0.13
	Quadratic	0.053869			
	Cubic	0.048465			
`mboost()`	Tuned d.f & knots	0.091156	10,000	42	3.73
	Tuned d.f	2.745459	10,000	17	2.85
Gamma					
`NGeDS()`	Linear	0.002359		13	0.15
	Quadratic	0.001676			
	Cubic	0.003808			
`NGeDSboost()`	Linear	0.063365	2	17.5	0.16
	Quadratic	0.001390			
	Cubic	0.001614			
`mboost()`	Tuned d.f & knots	0.027172	10,000	76	6.67
	Tuned d.f	0.027336	10,000	18	7.42
Example 6.2					
`NGeDS()`	Linear	0.021640		36	0.14
	Quadratic	0.022293			
	Cubic	0.035854			
`NGeDSboost()`	Linear	0.021375	3	38	0.18
	Quadratic	0.022165			
	Cubic	0.035695			
`mboost()`	Tuned d.f & knots	0.054078	10,000	92	1.29
	Tuned d.f	0.161881	10,000	38	2.12

Table 3: Median mean squared error/mean deviance, number of internal knots and computation time for `NGeDS()`, `NGeDSboost()` and `mboost()` for the fits over 1,000 simulated datasets for examples 6.1 and 6.2.

Table 3 displays the median MSE/mean deviance, median number of boosting iterations, median number of internal knots and median fitting time, for each of the models. Both `NGeDS()` and `NGeDSboost()` demonstrate high accuracy when considering the median MSEs/mean deviances obtained for each example. `NGeDSboost()` exhibits a slightly lower error compared to `NGeDS()`, with only a minor increase in computation time. The first implementation of `mboost()` that was considered fails to surpass the accuracy levels of the GeDS models, even after exhaustively tuning the number of knots and the degrees of freedom of the P-spline. In addition, GeDS models require a lower number of internal knots to achieve the median MSEs/mean deviances obtained, and fewer boosting iterations in the case of FGB-GeDS, hence demonstrating greater parametric efficiency. The accuracy of `mboost()` falls dramatically when setting the number of knots to be equal to the median number of knots of the final FGB-GeDS fits and only tuning the degrees of freedom.

In Figure 2, boxplots for the Normal, Poisson and Gamma version of example 6.1 are depicted. For ease of comparison, only the cubic GeDS and cubic FGB-GeDS models are included, which were the best-performing ones. As it can be observed, `NGeDSboost()` indeed boosts the performance of the already highly competitive canonical GeDS for example 6.1, showcasing notably higher resistance to outliers. The performance of `mboost()`, when tuning both number of knots and d.f., is fairly good for the Normal and Poisson case, though not better than FGB-GeDS. In the Gamma case, despite extensive tuning, `mboost()` still exhibits relatively high MSE values and an unstable performance, as evidenced by the wide interquartile range and long whiskers in the boxplot. Across all cases of example 6.1, the MSEs/mean deviances for `mboost()` are substantially higher when the number of knots is set to the median obtained for `NGeDSboost()`, with tuning applied only to the d.f.

For more detailed insight on example 6.2, Figure 3 presents the fits on the first simulated dataset. It can be observed that `mboost()` tends to overfit the data for the case where both the number of knots and the d.f. are tuned. Additionally, it fails to capture the origin of the Doppler function when the number of knots is fixed to the median used by `NGeDSboost()` and only the d.f. are tuned. In contrast, with a similar number of knots, the GeDS models provide a fairly good fit across the entire function range.

Next, a real data application is considered.

Example 6.3. - $\mathbf{BaFe_2As_2}$. Real data example with $N = 1,151$ from a superconductivity study of Barium-Ferrum-Arsenide ($BaFe_2As_2$) through a neutron diffraction experiment, carried out by Kimber, Kreyssig, Zhang, Jeschke, Valentí, Yokaichiya, Colombier, Yan, Hansen, Chatterji, McQueeney, Canfield, Goldman, and Argyriou (2009) and considered by Kaishev *et al.* (2016) and Dimitrova *et al.* (2023). Important information about the structural properties of the $BaFe_2As_2$ compound is retrieved by analyzing the position, height and width of the peaks in the data. For further details on this example see Kaishev *et al.* (2016).

```
R> # Example 6.3. - BaFe2As2
R> data('BaFe2As2')
R> data = BaFe2As2
R> Y <- data$intensity
R> X <- data$angle
R> data <- data.frame(X,Y)
```

For example 6.3, we set $\beta = 0.6$, $\phi = 0.995$ and $q = 3$, and keep the defaults for the rest of parameters. Given that the primary objective here is to efficiently capture the signal in the data without overfitting, we fix the number of knots in `mboost()` to match that of `NGeDSboost()` and tune only the d.f. Since the optimal GeDS and FGB-GeDS fit in this case is linear, the degree of the P-spline was also tuned; however, the default setting `degree=3` still yielded the best result for `mboost()`.

The models are compared according to their empirical MSE in Table 4, and plots are presented in Figure 4. **mboost** fails to effectively capture the intensity peaks in the $BaFe_2As_2$ dataset; this contrasts with the GeDS models which clearly allow for a more accurate estimation of the structural parameters of this material.

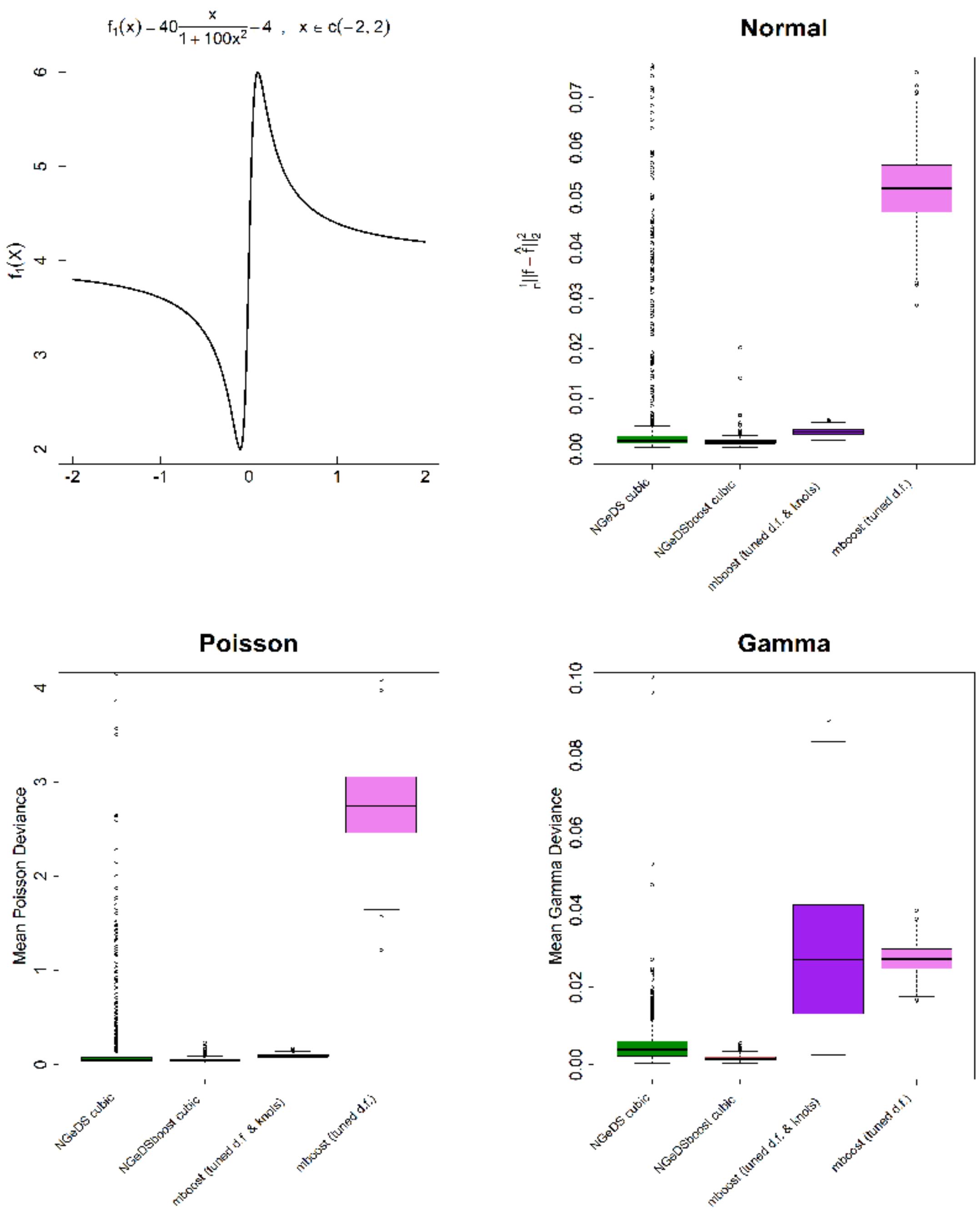


Figure 2: Cubic `NGeDS()`, cubic `NGeDSboost()` and `mboost()` fits over 1,000 different simulated datasets of example 6.1. The original function, $f_1(x)$, is depicted in black on the top left corner.

7. Numerical examples – generalized additive models

The two main packages in R for fitting generalized additive models are **gam** (Hastie (2024)) and **mgcv** (Wood (2023)). On the one hand, **gam** follows the theory outlined in Hastie and

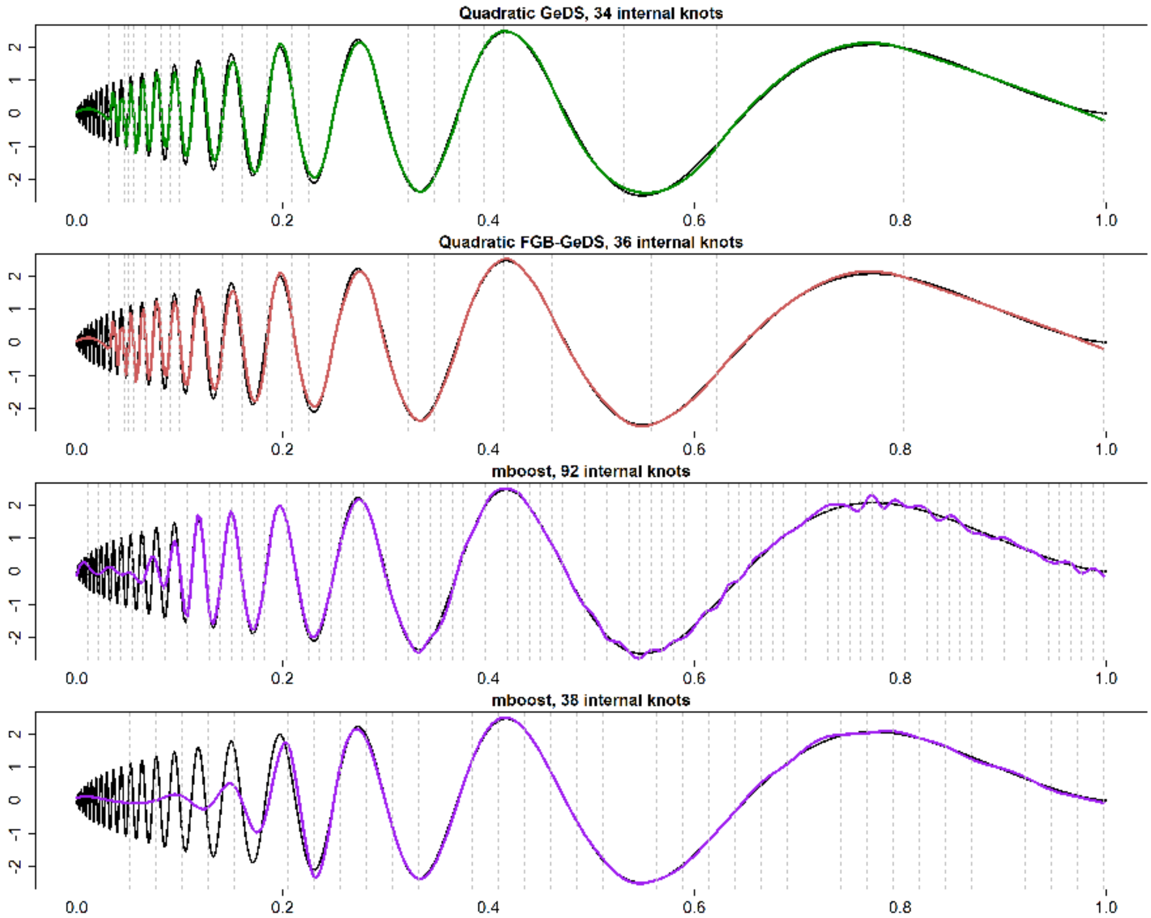


Figure 3: Quadratic NGeDS() and NGeDSboost() fits, and mboost() fits over a simulated dataset of example 6.2 - Doppler function. The original function, $f_2(x)$, is depicted in black. The dotted vertical lines denote the locations of the knots.

	Type	MSE	Boosting iter.	Internal knots	Time (sec.)
Example 6.3					
	Linear	59,385.34		282	12.19
NGeDS()	Quadratic	70,817.56			
	Cubic	103,247.37			
	Linear	59,370.17	3	284	11.64
NGeDSboost()	Quadratic	70,655.68			
	Cubic	103,079			
mboost()	Tuned d.f & knots	1,563,342.22	10,000	284	2.37

Table 4: Empirical mean squared error, number of internal knots and computation time for NGeDS(), NGeDSboost() and mboost() for the fits on the real data example 6.3.

Tibshirani's seminal work, Hastie and Tibshirani (1990). On the other hand, **mgcv**, by Simon Wood, is a package for estimating penalized generalized linear models—which include generalized additive models as a special case—and therefore generalizes Hastie and Tibshirani's approach. While **gam** employs backfitting and local scoring algorithms for combining smooth functions of the predictors, **mgcv** utilizes basis function expansions of the predictor functions,

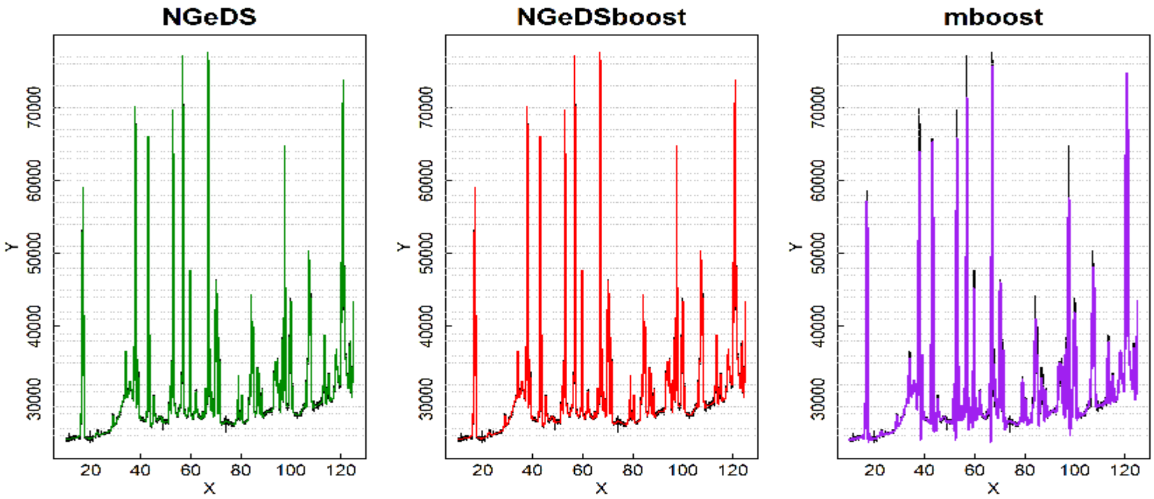


Figure 4: Linear `NGeDS()`, linear `NGeDSboost()` and `mboost()` fits over example 6.3 - $BaFe_2As_2$. The original data is depicted in black.

each with an associated penalty controlling function smoothness. Estimation is then be carried out by penalized regression methods, and the appropriate degree of smoothness for the f_j is estimated from data using cross validation or marginal likelihood maximization (Wood (2017)).

We compare the performance of the `NGeDSgam()` function—which implements Hastie and Tibshirani's approach using normal GeD splines, `NGeDS()`, as smoothers—with **gam** and **mgcv**, which respectively use smoothing splines and thin plate regression splines, both denoted by `s()`. This comparison is based on examples extracted from these packages. Given the more dispersed nature of the data in this examples, the parametrization of `NGeDSgam()` focused on avoiding overfitting by relaxing the stopping rule for the function smoothers (through lower values of ϕ and q).

7.1. Examples from gam package

Example 7.1 - airquality. The first example is based on the `airquality` dataset, which is available from base R. This presents daily air quality measurements in New York, from May 1, 1973, to September 30, 1973, and includes the following four variables:

- `Ozone`: Mean concentration of ozone, measured in parts per billion, recorded between 1:00 PM and 3:00 PM at Roosevelt Island.
- `Solar.R`: Solar radiation, expressed in *Langleys*, within the frequency band of 4,000 to 7,700 *Angstroms*. The data was collected between 8:00 AM and 12:00 PM at Central Park.
- `Wind`: Average of the wind speeds measured at 7:00 AM and 10:00 AM, reported in miles per hour, at LaGuardia Airport.
- `Temp`: Maximum daily temperature, in degrees Fahrenheit, at LaGuardia Airport.

The data were obtained from the New York State Department of Conservation (ozone data) and the National Weather Service (meteorological data). See Chambers (1983) for further reference.

We follow the implementation of this example as presented in Hastie (2024) and correspondingly adapt it for **mgcv** and **GeDS**. Note this is time series data. Thus, to compare the performance of the models, these are trained on earlier data and tested on more recent data, in order to simulate realistic forecasting scenarios. A sequence of train/test split ratios ranging from 60% to 90%, in 1% increments, is considered. The R code utilized is as follows:

```
R> # Example 7.1
R> library(gam)
R> data(airquality)
R> airquality <- na.omit(airquality)
R> airquality$Ozone <- airquality$Ozone^(1/3)
R> airquality <- airquality[order(airquality$Month, airquality$Day),]

R> trainIndex <- round(nrow(airquality) * ratio)
R> train <- airquality[1:trainIndex, ]
R> test <- airquality[(trainIndex + 1):nrow(airquality), ]

R> library(gam)
R> mod_gam <- gam(Ozone ~ lo(Solar.R) + lo(Wind,Temp), data = train)
R> detach(package:gam, unload = TRUE)

R> library(mgcv)
R> mod_mgcv <- gam(Ozone ~ s(Solar.R) + s(Wind,Temp), data = train)
R> detach(package:mgcv, unload = TRUE)

R> library(GeDS)
R> Gmodgam <- NGeDSgam(Ozone ~ f(Solar.R) + f(Wind, Temp), data = train,
+   phi = 0.7)
```

Note that in `NGeDSgam()`, a value of `phi = 0.7` (i.e., lower than the default 0.99) is set to avoid overfitting. The training and test MSEs are plotted in Figure 5, where it can be seen that the linear `NGeDSgam()` consistently outperforms the other models on the test set.

Examples 7.2 and 7.3 - Kyphosis. The third and fourth examples from the **gam** package are based on the `kyphosis` dataset. This dataset presents the outcomes of corrective spinal surgery (specifically, "laminectomies") performed on children to correct a condition known as "kyphosis" (a spinal deformation; see Hastie and Tibshirani (1990) for details). It comprises 81 observations and includes the following four variables:

- `Kyphosis`: A response factor with levels "absent" or "present", indicating whether kyphosis was present after the operation.
- `Age`: The age of the child in months, represented as a numeric vector.
- `Number`: The number of vertebrae involved in the operation, also a numeric vector.

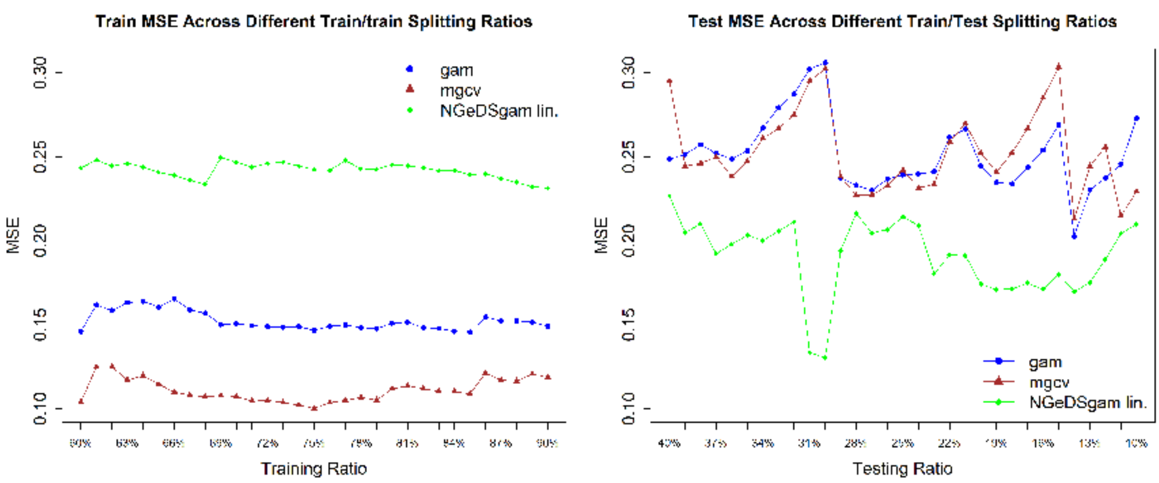


Figure 5: Train (on the left) and test (on the right) MSE across various train/test sample splitting ratios for Example 7.1. Models compared are **gam**, **mgcv** and linear `NGeDSgam()`.

- `Start`: The number of the first (topmost) vertebra operated on, also a numeric vector.

The objective is to determine the presence of kyphosis using the information provided by the features. In example 7.2 the covariates considered are `Number` and `Age`, while in 7.3, `Age` and `Start` are considered instead. The models are compared according to the test MSE and different train/test splitting ratios were considered: 65%/35%, 70%/30% and 75%/25%. Besides, 100 different splits are simulated for each train/test splitting ratio. The R implementation of these examples is as follows:

```
R> # Example 7.2
R> set.seed(123)
R> n <- nrow(kyphosis)
R> trainIndex <- sample(1:n, size = floor(training_ratio * n))
R> train <- kyphosis[trainIndex, ]
R> test <- kyphosis[-trainIndex, ]

R> library(gam)
R> mod_gam <- gam(Kyphosis ~ Number + s(Age,4), family = binomial,
+   data = train)
R> detach(package:gam, unload = TRUE)

R> library(mgcv)
R> mod_mgcv <- gam(Kyphosis ~ Number + s(Age), family = binomial, data = train)
R> detach(package:mgcv, unload=TRUE)

R> library(GeDS)
R> Gmodgam <- NGeDSgam(Kyphosis ~ Number + f(Age), data = train,
+   family = binomial, phi = 0.7)
```

```
R> # Example 7.3
R> kyphosis <- subset(kyphosis, Number > 2)
R> set.seed(123)
R> n <- nrow(kyphosis)
R> trainIndex <- sample(1:n, size = floor(training_ratio * n))
R> train <- kyphosis[trainIndex, ]
R> test <- kyphosis[-trainIndex, ]

R> library(gam)
R> mod_gam <- gam(Kyphosis ~ poly(Age,2) + s(Start), family = binomial,
+   data = train)
R> detach(package:gam, unload = TRUE)

R> library(mgcv)
R> mod_mgcv <- gam(Kyphosis ~ s(Age) + s(Start), family = binomial,
+   data = train)

R> library(GeDS)
R> Gmodgam <- NGeDSgam(Kyphosis ~ f(Age) + f(Start), data = train,
+   family = binomial, q_gam = 1, phi = 0.7),
```

Again, to avoid overfitting we define a "weaker" function smoother through `phi = 0.7` and relax the local-scoring stopping rule setting `q_gam = 1`, for both for 7.2 and 7.3. Figure 6 shows boxplots of the binomial deviance on the test set, illustrating the more stable performance of linear `NGeDSgam` compared to **gam** and **mgcv**, as evidenced by its tighter interquartile range across the different train/test splitting ratios considered.

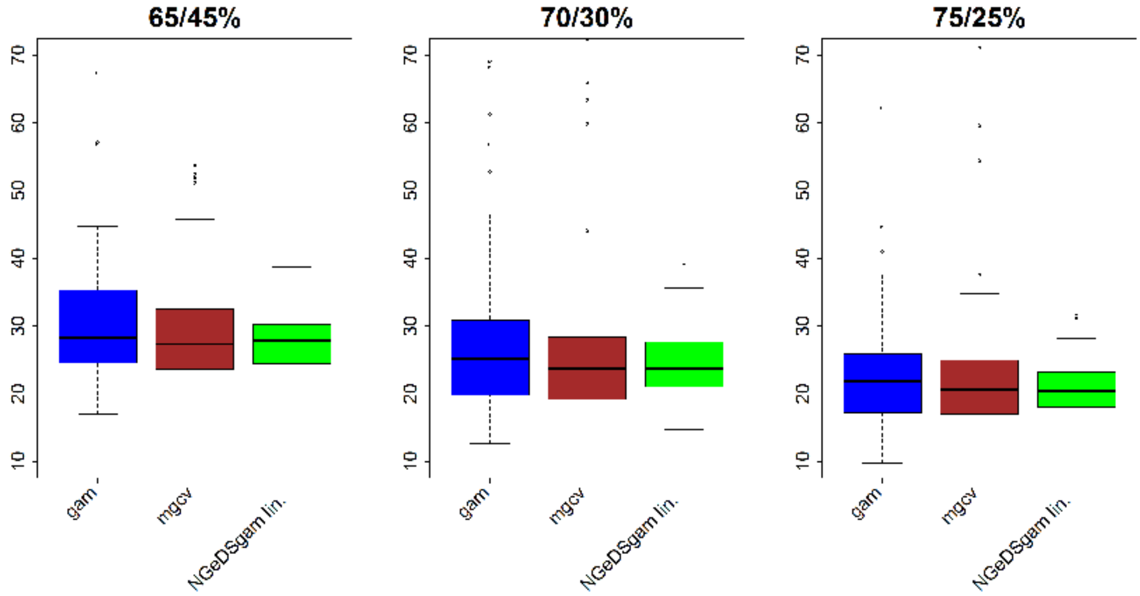
7.2. Examples from mgcv package

We now consider two examples based on datasets generated with the `gamSim()` function from the **mgcv** package. The aim of this function is to simulate data that allows to illustrate the use of the `mgcv::gam()` function. First, in 7.4, we explore an example from Gu and Wahba (1991) that involves four uniform covariate terms. The true function is defined by $f(\mathbf{x}) = 2 \times \sin(\pi \times x_0) + \exp(2x_1) + 0.2x_2^{11}(10(1-x_2))^6 + 10(10x_2)^3(1-x_2)^{10}$. However, we consider $y = f(\mathbf{x}) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$, to be the observed response variable and also include a noise predictor, x_3, as part of the covariates of the model. Second, in 7.5, instead of having $\sin(\pi \times x_0)$, a factor variable x_0 that comprises four categories is included, and the cases with and without the noise predictor x_3 are considered. Consistent with our previous criteria, the MSEs are calculated with respect to the true function. For each example, 1,000 different datasets were simulated.

Example 7.4 - Gu and Wahba 4 covariate term example.

```
R> set.seed(123)
R> f_x0x1x2 <- function(x0,x1,x2) {
R>   f0 <- function(x0) 2 * sin(pi * x0)
R>   f1 <- function(x1) exp(2 * x1)
```

(a) Example 7.2.



(b) Example 7.3.

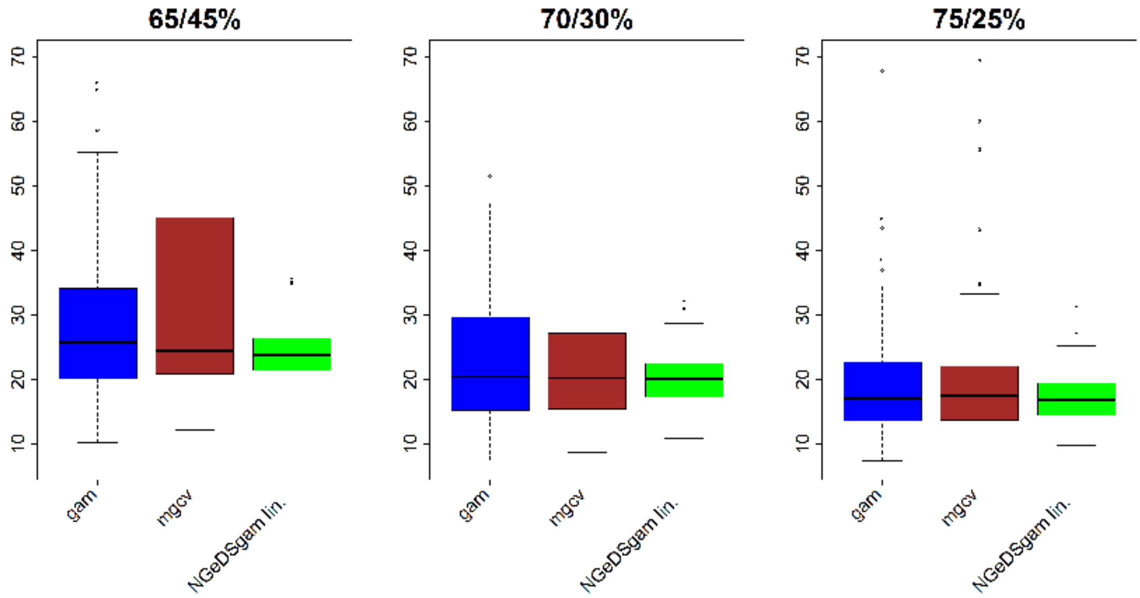


Figure 6: Test binomial mean deviance for 100 simulated train/test data sets from Examples 7.2 and 7.3, for different train/test splitting ratios (65/45%, 70/30% and 75/25%). Models compared are **gam**, **mgcv** and linear `NGeDSgam()`.

```
R>   f2 <- function(x2) {
R>     0.2 * x2^11 * (10 * (1 - x2))^6 + 10 * (10 * x2)^3 * (1 - x2)^10
R>     }
R>   f <- f0(x0) + f1(x1) + f2(x2)
```

```
R>   return(f)
R> }
R> library(mgcv)
R> data <- gamSim(eg = 1, n = 400, dist = "normal", scale = 0.2)
R> f <- f_x0x1x2(x0 = data$x0, x1 = data$x1, x2 = data$x2)
R> detach(package:mgcv, unload=TRUE)

R> library(gam)
R> gam_mod <- gam(y ~ s(x0) + s(x1) + s(x2) + s(x3), data = data)
R> detach(package:gam, unload = TRUE)
R> library(mgcv)
R> mod_mgcv <- gam(y ~ s(x0) + s(x1) + s(x2) + s(x3), data = data)
R> detach(package:mgcv, unload = TRUE)
R> library(GeDS)
R> Gmodgam <- NGeDSgam(y ~ f(x0) + f(x1) + f(x2) + f(x3), data = data)
```

Example 7.5 - An additive example plus a factor variable.

```
R> set.seed(123)
R> f_x0x1x2_factor <- function(x0,x1,x2) {
R>   f0 <- function(x0) 2 * as.numeric(x0)
R>   f1 <- function(x1) exp(2 * x1)
R>   f2 <- function(x2) {
R>     0.2 * x2^11 * (10 * (1 - x2))^6 + 10 * (10 * x2)^3 * (1 - x2)^10
R>   }
R>   f <- f0(x0) + f1(x1) + f2(x2)
R>   return(f)
R> }
R> library(mgcv)
R> data = gamSim(eg = 5, n = 200, dist = "normal", scale = 0.2)
R> f <- f_x0x1x2_factor(x0 = data$x0, x1 = data$x1, x2 = data$x2)
R> detach(package:mgcv, unload=TRUE)

R> library(gam)
R> gam_mod_1 <- gam(y ~ x0 + s(x1) + s(x2) + s(x3), data = data)
R> gam_mod_2 <- gam(y ~ x0 + s(x1) + s(x2), data = data)
R> detach(package:gam, unload = TRUE)
R> library(mgcv)
R> mod_mgcv1 <- gam(y ~ x0 + s(x1) + s(x2) + s(x3), data = data)
R> mod_mgcv2 <- gam(y ~ x0 + s(x1) + s(x2), data = data)
R> detach(package:mgcv, unload = TRUE)
R> library(GeDS)
R> Gmodgam_1 <- NGeDSgam(y ~ x0 + f(x1) + f(x2) + f(x3), data = data)
R> Gmodgam_2 <- NGeDSgam(y ~ x0 + f(x1) + f(x2), data = data)
```

For the three examples—7.4, 7.5.1 and 7.5.2—, `NGeDSgam()` was run with its default parameters. Figure 8 presents boxplots of the MSEs obtained based on 1,000 simulations for each

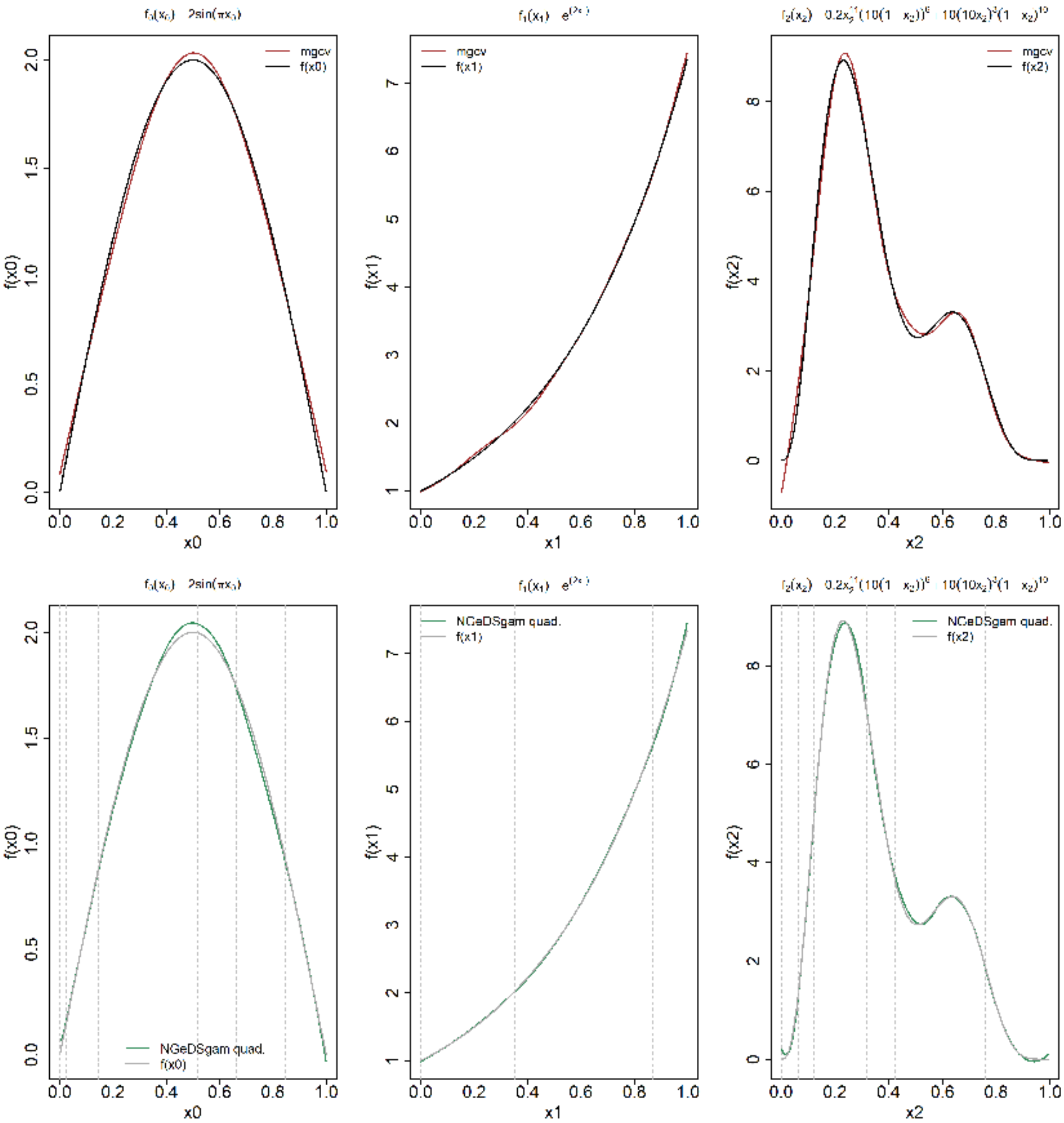


Figure 7: Partial `mgcv::gam()` and quadratic GeDS-GAM fits on each of the component functions, $f(x_0)$, $f(x_1)$ and $f(x_2)$, that additively form $f(\mathbf{x})$, based on a simulated data sample from Example 7.4. The dotted vertical lines in the plots below denote the locations of the knots in the GAM-GeDS fit.

example. To facilitate a closer comparison between **mgcv** and `NGeDSgam()`, **gam** was excluded due to its consistently poor performance. By a significant margin, both quadratic and cubic `NGeDSgam()` show superior accuracy. Figure 7 illustrates the partial fits for one of these simulations using `mgcv::gam()`, and using quadratic GeDS-GAM. While both models effectively recover $f(x_0)$ and $f(x_1)$, GeDS-GAM outperforms in capturing the more intricate structure of $f(x_2)$, which suggests that the accuracy difference observed in the boxplots may stem from

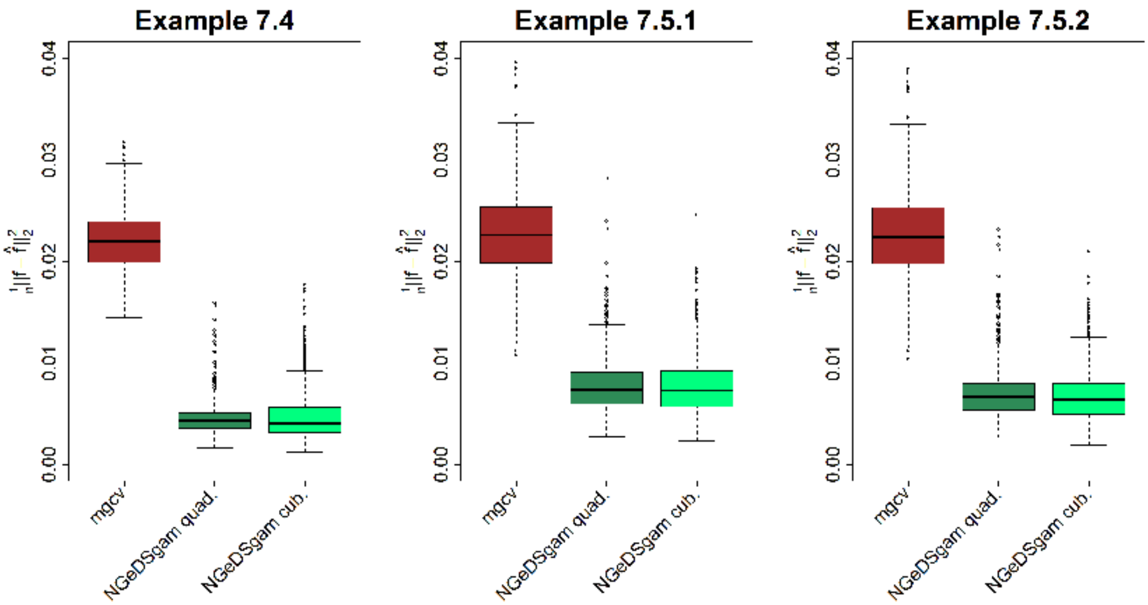


Figure 8: Boxplots of the MSE for **mgcv**, and for quadratic and cubic `NGeDSgam()` fits based on 1,000 simulations from Examples 7.4 and 7.5. MSE is calculated with respect to the simulated data without noise, i.e., the true generating function.

GeDS-GAM's better handling more complex structures.

8. Numerical examples – component-wise boosted models

Bühlmann and Yu (2003) demonstrate that their L_2Boosting approach competes effectively with standard estimation techniques for fitting additive models, such as backfitting, and can even surpass their performance. We now present two examples included in the **mboost** documentation (Hothorn *et al.* (2022)) and observe how, after some parameter adjustment, `NGeDSboost()` performs on par with, or even better than, **mboost**. Additionally, we present a high-dimensional binary classification problem included in Bühlmann (2006) and Bühlmann and Hothorn (2007).

Example 8.1 - Bodyfat. The first example is based on `bodyfat` dataset (Garcia, Wagner, Hothorn, Koebnick, Zunft, and Trippo (2005)) which collects observations of body fat, age and eight different anthropometric measurements for 71 German women. The goal is to accurately predict women's body fat using the available anthropometric measurements, since direct measurements are frequently too expensive. The model specification presented in **mboost** documentation utilizes a combination of a linear base-learner `bols(age)`, a stump-based base-learner `btree(hipcirc, waistcirc)` (hip and waist circumference, respectively), and a P-spline learner `bbs(kneebreadth)` (breadth of the knee). The response variable, `DEXfat`, corresponds to Dual X-Ray Absorptiometry (DXA) body fat measurements.

```
R> # Example 8.1 - Bodyfat
R> library(mboost)
R> data("bodyfat", package = "TH.data")
```

```
R> n <- nrow(bodyfat)
R> set.seed(123)
R> # Create a random sample of row indices for the training set
R> trainIndex <- sample(1:n, size = floor(ratio * n))
R> # Subset the data into training and test sets
R> train <- bodyfat[trainIndex, ]
R> test <- bodyfat[-trainIndex, ]

R> ### model conditional expectation of DEXfat given
R> mod_mboost <- mboost(DEXfat ~
+   bols(age) + ### a linear function of age
+   btree(hipcirc, waistcirc) + ### a smooth non-linear interaction of
+   ### hip and waist circumference
+   bbs(kneebreadth), ### a smooth function of kneebreadth
+   data = train, control = boost_control(mstop = 1000, nu = 0.1))

R> library(GeDS)
R> Gmodboost <- NGeDSboost(formula = DEXfat ~ age + f(hipcirc, waistcirc) +
+   f(kneebreadth), data = train, initial_learner = FALSE, shrinkage = 0.6,
+   phi = 0.9, q = 1, higher_order = FALSE)
```

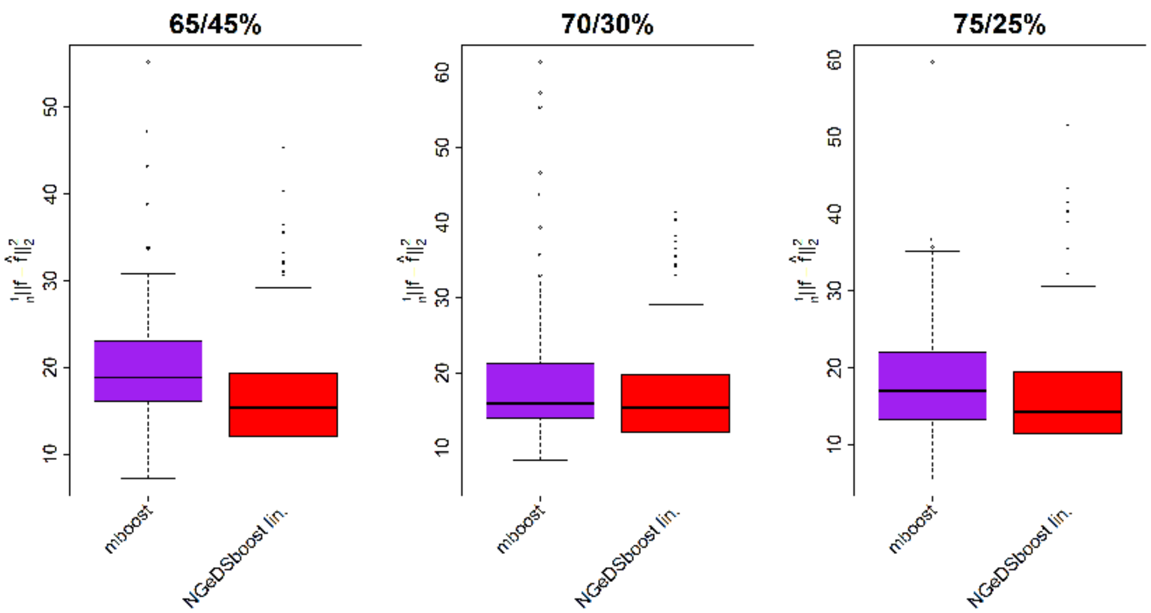


Figure 9: Example 8.1. Test MSE for 100 simulated train/test data splits for different train/test splitting ratios (65/45%, 70/30% and 75/25%). Models compared are `mboost()` and linear `NGeDSboost()` models.

By setting `initial_learner = FALSE` in `NGeDSboost()` the corresponding empirical risk minimizer is utilized as initial learner, instead of a GeD spline fit. The strength of the `NGeDS()` base-learners at each boosting iteration is diminished setting `phi = 0.9`, `q = 1`, and `shrinkage = 0.6`; `higher_order = FALSE`, since only the linear `NGeDSboost()` fit is used, and hence there is no need of computing the quadratic and cubic fits. The performance of the models is compared based on the test MSE for different train/test splitting ratios: 65/35%, 70/30%, and 75/25%. For each splitting ratio, 100 different splits are simulated. Under the suggested parametrization, linear `NGeDSboost()` performs slightly better than `mboost()` for each of the train/test splits.

Example 8.2 - Synthetic data w/4 predictors. The second example consists of a synthetic dataset of 100 observations with four predictors: two are continuous (x_1 and x_2), one is binary (x_3), and one is multi-categorical (x_4). The response variable is $y = f(\mathbf{x}) + \epsilon$, where $f(\mathbf{x}) = 3\sin(x_1) + (x_2)^2$ and $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$. Additionally, observation weights are predefined. For the `mboost()` function, a vector defining the positions of the interior knots is also provided, and the model specification presented by the package documentation consists of a combination of P-splines base-learners, `bbs(...)`, and categorical effects, wrapped in `bols(...)`.

```
R> # Example 8.2 - Synthetic data w/4 predictors
R> library(mboost)
R> set.seed(290875)
R> n <- 100
R> x1 <- rnorm(n)
R> x2 <- rnorm(n) + 0.25 * x1
R> x3 <- as.factor(sample(0:1, 100, replace = TRUE))
R> x4 <- gl(4, 25)
R> y <- 3 * sin(x1) + x2^2 + rnorm(n)
R> weights <- drop(rmultinom(1, n, rep.int(1, n) / n))
R> knots.x2 <- quantile(x2, c(0.25, 0.5, 0.75))
R> ### more convenient formula interface
R> mod_mboost <- mboost(y ~ bbs(x1, knots = 20, df = 4) +
 bbs(x2, knots = knots.x2, df = 5) +
 bols(x3) + bols(x4), weights = weights)

R> library(GeDS)
R> Gmodboost <- NGeDSboost(formula = y ~ f(x1) + f(x2) + x3 + x4,
+   data = data, weights = weights, phi = 0.9)
```

MSEs are calculated with respect to the true function $f(\mathbf{x})$, i.e. excluding the random normal noise that is added on y. As shown in the left panel of Figure 10, simply setting a lower `phi` to reduce the strength of the GeDS base-learners, both the quadratic and cubic versions of `NGeDSboost()` improve the performance of `mboost()`.

Example 8.3. - Breast Cancer Gene Expression. Variable selection is especially important in high-dimensional situations. Bühlmann (2006) and Bühlmann and Hothorn (2007) study a binary classification problem involving $P = 7,129$ gene expression levels in $N = 49$ breast cancer tumor samples (data taken from West, Blanchette, Dressman, Huang, Ishida, Spang, Zuzan, Olson, Marks, and Nevins (2001)). For each observation, a binary response variable, `nodal.y`, describes the lymph node status (25 are negative and 24 are positive). The code implementation by Bühlmann and Hothorn (2007) uses linear models as base-learners and is as follows:

```
R> data("Westbc", package = "TH.data")
R> data <- data.frame(Westbc$pheno, t(Westbc$assay))
R> mod_mboost <- glmboost(nodal.y ~ ., data = data,
+   family = Binomial(link = c("logit")),
+   control = boost_control(mstop = 200, center = TRUE))
```

However, let us note that when trying to run a similar model but using P-splines as base-learners an error is displayed and the R session is aborted.

```
R> var_names <- names(data)
R> var_names <- var_names[var_names != "nodal.y"]
R> var_names <- paste0("bbs(", var_names, ",knots = 20, degree = 1,
+   center = FALSE)")
R> formula <- paste("nodal.y ~", paste(var_names, collapse = " + "))
R> mod_mboost <- gamboost(as.formula(formula), data = data,
+   family = Binomial(link = c("logit")),
+   control = boost_control(mstop = 200, center = TRUE))
```

```
Error: C stack usage  15924464 is too close to the limit
```

This is not the case for `NGeDSboost()`, using GeD splines base-learners:

```
R> Gmodboost <- NGeDSboost(formula = nodal.y ~ ., data = data,
+   family = mboost::Binomial(), initial_learner = FALSE, shrinkage = 0.1,
+   phi_boost_exit = 0.95, phi = 0.4, higher_order = FALSE)
```

Mean binomial test deviance boxplots based on 100 simulated data splits for a train/test split of 70%/30% are included on the right panel of Figure 10. As observed, linear `NGeDSboost()`, while evidently more computationally expensive, clearly outperforms the `glmboost()` implementation in terms of accuracy. Similar to previous examples, the strength of the GeDS base-learner is reduced setting `shrinkage = 0.1` and `phi = 0.4`, while a looser stopping rule is employed by fixing `phi_boost_exit = 0.95`.

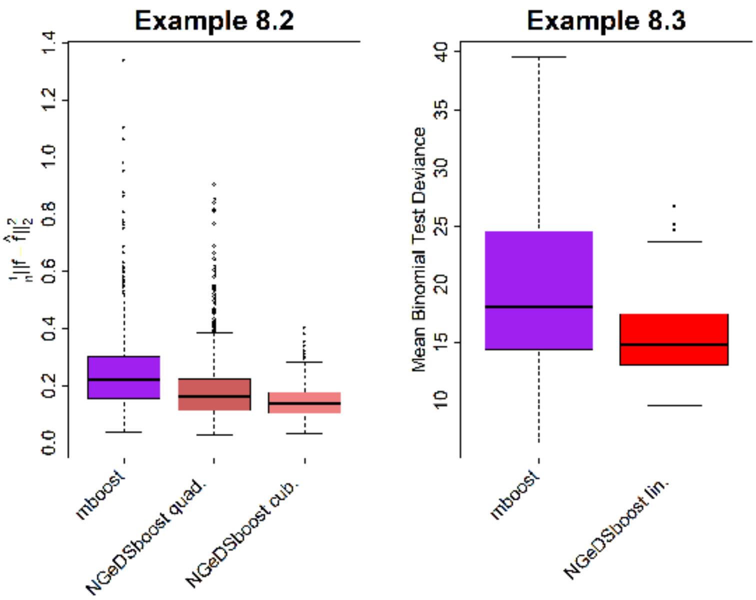


Figure 10: *(Left)* MSE boxplots for `mboost()` and `NGeDSboost()` fits on 1,000 simulated data samples of example 8.2. *(Right)* Boxplots of the mean binomial test deviance for `glmboost()` and linear `NGeDSboost()` fits on 100 simulated data splits of example 8.3, using a training/test ratio of 70%/30%.

9. Conclusions

In this paper, the GeDS methodology, introduced by Kaishev *et al.* (2016) and Dimitrova *et al.* (2023), is significantly enhanced through its extension towards the benchmark of generalized additive models (GAM) and functional gradient boosting (FGB). In addition, the R package **GeDS**, which implements these methods, is introduced. The main contributions of this work are summarized as follows.

Firstly, we develop a new method for functional gradient boosting based on the use of GeD splines as base-learners. Unlike its competitors, the final boosted fit is explicitly expressed as a single spline model (i.e., not just as a sum of learners), thereby enhancing the model's transparency, interpretability and flexibility. Additionally, the number of boosting iterations is controlled by a simple stopping rule, avoiding the complexities associated with cross-validation or AIC-based techniques. Within the realm of non-additive models, this method not only outperforms its direct competitors but also enhances the already robust canonical GeDS implementation. Secondly, we have extended the GeDS methodology to the truly multivariate case in two different ways. On the one hand, through its incorporation into generalized additive models, by using GeD splines as function smoothers within the local scoring algorithm. On the other hand, through component-wise gradient boosting.

The GeDS, GAM-GeDS, and FGB-GeDS models offer unparalleled versatility in statistical fitting, providing simultaneous linear, quadratic, and cubic spline fits. This allows users to select the degree of the final fit, balancing smoothness and accuracy. Each model can be finely tuned via two primary parameters, ϕ and β, with the addition of ϕ_{gam}^{exit} in FGB-GAM and ϕ_{boost}^{exit} and $\kappa_0^{\max}$ in FGB-GeDS. The three approaches stand out for their efficient and highly accurate fits, effectively handling both smooth and complex, wiggly univariate functions, as well as multivariate additive problems, regardless of their dimension. Overall, quadratic and cubic GeDS models demonstrate a strong ability to accurately fit intricate functions, which constitutes a significant edge over their competitors. Meanwhile, in sparser data settings and high-dimensional contexts, linear GeDS is more appropriate.

In conclusion, GeDS methodology—originally developed for the Normal univariate case and later extended to the GNM framework—successfully extends to the context of generalized additive models and functional gradient boosting, demonstrating competitive performance in terms of accuracy and efficiency, along with compelling structural properties when compared to existing methods.

In ongoing research, two immediate extensions of GeDS have been identified: quantile regression and varying coefficient models. First, in settings with a continuous response, quantile regression enables modeling of various conditional quantiles (Koenker (2005)). Second, varying coefficient models (Hastie and Tibshirani (1993)) allow regression coefficients to vary systematically and smoothly across multiple dimensions, and thus to deal with, e.g., time series data.

Computational details

The results in this paper, displayed in Sections 6, 7 and 8, were obtained using R 4.4.2 on a standard PC (Intel(R) Core(TM) Ultra 9 185H, 2.50 GHz, 64 GB RAM). Main packages used are **GeDS** 0.3.0, **mboost** 2.9.11, **gam** 1.22.5 and **mgcv** 1.9.1. R itself and all packages used are available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/`.

References

Bissantz N, Hohage T, Munk A, Ruymgaart F (2007). "Convergence Rates of General Regularization Methods for Statistical Inverse Problems and Applications." *SIAM Journal on Numerical Analysis*, **45**(6), 2610–2636. doi:10.1137/060651884. https://doi.org/10.1137/060651884, URL https://doi.org/10.1137/060651884.

Breiman L (1998). "Arcing classifier (with discussion and a rejoinder by the author)." *The Annals of Statistics*, **26**(3), 801 – 849. doi:10.1214/aos/1024691079. URL https://doi.org/10.1214/aos/1024691079.

Breiman L (1999). "Prediction Games and Arcing Algorithms." *Neural Computation*, **11**(7), 1493–1517. ISSN 0899-7667. doi:10.1162/089976699300016106. https://direct.mit.edu/neco/article-pdf/11/7/1493/814214/089976699300016106.pdf, URL https://doi.org/10.1162/089976699300016106.

Bühlmann P (2006). "Boosting for high-dimensional linear models." *The Annals of Statistics*, **34**(2), 559 – 583. doi:10.1214/009053606000000092. URL https://doi.org/10.1214/009053606000000092.

Bühlmann P, Hothorn T (2007). "Boosting Algorithms: Regularization, Prediction and Model Fitting." *Statistical Science*, **22**(4), 477 – 505. doi:10.1214/07-STS242. URL https://doi.org/10.1214/07-STS242.

Bühlmann P, Van De Geer S (2011). *Statistics for high-dimensional data: methods, theory and applications*. Springer Science & Business Media.

Buja A, Hastie T, Tibshirani R (1989). "Linear Smoothers and Additive Models." *The Annals of Statistics*, **17**(2), 453 – 510. doi:10.1214/aos/1176347115. URL https://doi.org/10.1214/aos/1176347115.

Bühlmann P, Yu B (2003). "Boosting With the L2 Loss." *Journal of the American Statistical Association*, **98**(462), 324–339. doi:10.1198/016214503000125. URL https://doi.org/10.1198/016214503000125.

Chambers JM (1983). *Graphical Methods for Data Analysis*. 1 edition. Chapman and Hall/CRC. doi:10.1201/9781351072304. URL https://doi.org/10.1201/9781351072304.

Dimitrova DS, Kaishev VK, Lattuada A, Verrall RJ (2023). "Geometrically designed variable knot splines in generalized (non-)linear models." *Applied Mathematics and Computation*, **436**. ISSN 0096-3003. doi:10.1016/j.amc.2022.127493. © 2022. This manuscript version is made available under the CC-BY-NC-ND 4.0 license https://creativecommons.org/licenses/by-nc-nd/4.0/, URL https://www.sciencedirect.com/science/article/pii/S0096300322005677.

Freund Y, Schapire RE (1996). "Experiments with a New Boosting Algorithm." In *International Conference on Machine Learning*.

Freund Y, Schapire RE (1997). "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting." *Journal of Computer and System Sciences*, **55**(1), 119–139. ISSN 0022-0000. doi:10.1006/jcss.1997.1504. URL https://www.sciencedirect.com/science/article/pii/S002200009791504X.

Friedman J, Hastie T, Tibshirani R (2000). "Additive logistic regression: a statistical view of boosting (With discussion and a rejoinder by the authors)." *The Annals of Statistics*, **28**(2), 337 – 407. doi:10.1214/aos/1016218223. URL https://doi.org/10.1214/aos/1016218223.

Friedman JH (2001). "Greedy function approximation: A gradient boosting machine." *The Annals of Statistics*, **29**(5), 1189 – 1232. doi:10.1214/aos/1013203451. URL https://doi.org/10.1214/aos/1013203451.

Garcia AL, Wagner K, Hothorn T, Koebnick C, Zunft HJF, Trippo U (2005). "Improved prediction of body fat by measuring skinfold thickness, circumferences, and bone breadths." *Obesity research*, **13**(3), 626–634. doi:10.1038/oby.2005.67. URL http://www.ncbi.nlm.nih.gov/pubmed/15833949.

Gu C, Wahba G (1991). "Minimizing GCV/GML Scores with Multiple Smoothing Parameters via the Newton Method." *SIAM J. Sci. Comput.*, **12**, 383–398. URL https://api.semanticscholar.org/CorpusID:5789455.

Hastie T (2007). "Comment: Boosting Algorithms: Regularization, Prediction and Model Fitting." *Statistical Science*, **22**(4), 513–515. ISSN 08834237. URL http://www.jstor.org/stable/27645856.

Hastie T (2024). *gam: Generalized Additive Models*. R package version 1.22-5, URL https://CRAN.R-project.org/package=gam.

Hastie T, Tibshirani R (1986). "Generalized Additive Models." *Statistical Science*, **1**(3), 297–310. ISSN 08834237. URL http://www.jstor.org/stable/2245459.

Hastie T, Tibshirani R (1990). "Generalized additive models." *Monographs on statistics and applied probability. Chapman & Hall*, **43**, 335.

Hastie T, Tibshirani R (1993). "Varying-Coefficient Models." *Journal of the Royal Statistical Society. Series B (Methodological)*, **55**(4), 757–796. ISSN 00359246. URL http://www.jstor.org/stable/2345993.

Hastie T, Tibshirani R, Friedman JH, Friedman JH (2009). *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer.

Hofner B, Mayr A, Robinzonov N, Schmid M (2014). "Model-based boosting in R: a hands-on tutorial using the R package mboost." *Computational Statistics*, **29**(1), 3–35.

Hofner B, Mayr A, Schmid M (2016). "gamboostLSS: An R Package for Model Building and Variable Selection in the GAMLSS Framework." *Journal of Statistical Software*, **74**(1), 1–31. doi:10.18637/jss.v074.i01. URL https://www.jstatsoft.org/index.php/jss/article/view/v074i01.

Hothorn T, Buehlmann P, Kneib T, Schmid M, Hofner B (2022). *mboost: Model-Based Boosting*. R package version 2.9-7, URL https://CRAN.R-project.org/package=mboost.

Jupp DLB (1978). "Approximation to Data by Splines with Free Knots." *SIAM Journal on Numerical Analysis*, **15**(2), 328–343. doi:10.1137/0715022. https://doi.org/10.1137/0715022, URL https://doi.org/10.1137/0715022.

Kaishev VK, Dimitrova DS, Haberman S, Verrall RJ (2016). "Geometrically designed, variable knot regression splines." *Computational Statistics*, **31**(3), 1079–1105. ISSN 1613-9658. doi:10.1007/s00180-015-0621-7. URL https://doi.org/10.1007/s00180-015-0621-7.

Karunasingha DSK (2022). "Root mean square error or mean absolute error? Use their ratio as well." *Information Sciences*, **585**, 609–629. ISSN 0020-0255. doi:https://doi.org/10.1016/j.ins.2021.11.036. URL https://www.sciencedirect.com/science/article/pii/S0020025521011567.

Kauermann G, Opsomer JD (2003). "Local likelihood estimation in generalized additive models." *Scandinavian Journal of Statistics*, **30**(2), 317–337.

Kimber SAJ, Kreyssig A, Zhang YZ, Jeschke HO, Valentí R, Yokaichiya F, Colombier E, Yan J, Hansen TC, Chatterji T, McQueeney RJ, Canfield PC, Goldman AI, Argyriou DN (2009). "Similarities between structural distortions under pressure and chemical doping in superconducting BaFe2As2." *Nature Materials*, **8**(6), 471–475.

Koenker R (2005). *Quantile Regression*. Econometric Society Monographs. Cambridge University Press.

Liang T, Sur P (2022). "A precise high-dimensional asymptotic theory for boosting and minimum-ℓ_1-norm interpolated classifiers." *The Annals of Statistics*, **50**(3), 1669 – 1695. doi:10.1214/22-AOS2170. URL https://doi.org/10.1214/22-AOS2170.

Mayr A, Binder H, Gefeller O, Schmid M (2014). "The evolution of boosting algorithms. From machine learning to statistical modelling." *Methods of information in medicine*, **53**(6), 419–427. doi:10.3414/ME13-01-0122. URL https://doi.org/10.3414/ME13-01-0122.

Mayr A, Hofner B, Schmid M (2012). "The importance of knowing when to stop. A sequential stopping rule for component-wise gradient boosting." *Methods of information in medicine*, **51**(2), 178–186. doi:10.3414/ME11-02-0030. URL https://doi.org/10.3414/ME11-02-0030.

Opsomer JD (2000). "Asymptotic Properties of Backfitting Estimators." *Journal of Multivariate Analysis*, **73**(2), 166–179. ISSN 0047-259X. doi:https://doi.org/10.1006/jmva.1999.1868. URL https://www.sciencedirect.com/science/article/pii/S0047259X99918687.

Potts S, Bergherr E, Reinke C, Griesbach C (2023). "Prediction-based Variable Selection for Component-wise Gradient Boosting." 2302.13822.

SAS Institute Inc (2018). *SAS/STAT® 15.1 User's Guide*. Cary, NC.

Schapire RE (1990). "The strength of weak learnability." *Machine Learning*, **5**(2), 197–227.

Schmid M, Hothorn T (2008a). "Boosting additive models using component-wise P-Splines." *Computational Statistics & Data Analysis*, **53**(2), 298–311. ISSN 0167-9473. doi:https://doi.org/10.1016/j.csda.2008.09.009. URL https://www.sciencedirect.com/science/article/pii/S0167947308004416.

Schmid M, Hothorn T (2008b). "Flexible boosting of accelerated failure time models." *BMC Bioinformatics*, **9**(1), 269. ISSN 1471-2105. doi:10.1186/1471-2105-9-269. URL https://doi.org/10.1186/1471-2105-9-269.

West M, Blanchette C, Dressman H, Huang E, Ishida S, Spang R, Zuzan H, Olson Jr JA, Marks JR, Nevins JR (2001). "Predicting the clinical status of human breast cancer by using gene expression profiles." *Proc Natl Acad Sci U S A*, **98**(20), 11462–11467.

Wood S (2017). *Generalized Additive Models: An Introduction with R*. 2 edition. Chapman and Hall/CRC. ISBN 9781315370279. URL https://doi.org/10.1201/9781315370279.

Wood SN (2023). *mgcv: Mixed GAM Computation Vehicle with Automatic Smoothness Estimation*. R package version 1.9-0, URL https://CRAN.R-project.org/package=mgcv.

Yang L, Hong Y (2017). "Adaptive penalized splines for data smoothing." *Computational Statistics & Data Analysis*, **108**, 70–83. ISSN 0167-9473. doi:https://doi.org/10.1016/j.csda.2016.10.022. URL https://www.sciencedirect.com/science/article/pii/S0167947316302493.

Yao Y, Rosasco L, Caponnetto A (2007). "On Early Stopping in Gradient Descent Learning." *Constructive Approximation*, **26**(2), 289–315. ISSN 1432-0940. doi:10.1007/s00365-006-0663-2. URL https://doi.org/10.1007/s00365-006-0663-2.

Yousuf K, Ng S (2021). "Boosting high dimensional predictive regressions with time varying parameters." *Journal of Econometrics*, **224**(1), 60–87. ISSN 0304-4076. doi:https://doi.org/10.1016/j.jeconom.2020.08.003. Annals Issue: PI Day, URL https://www.sciencedirect.com/science/article/pii/S0304407620302827.

Zhou S, Shen X (2001). "Spatially Adaptive Regression Splines and Accurate Knot Selection Schemes." *Journal of the American Statistical Association*, **96**(453), 247–259. ISSN 01621459. URL http://www.jstor.org/stable/2670363.

A. GeDS Boost model update

In Algorithm 4 we have presented the FGB-GeDS procedure using the L_2 loss. As mentioned, one of the main advantages of this technique is the possibility of updating the piecewise polynomial representation of the univariate base-learner(s) at each boosting iteration. This allows to express the final boosted model as a single spline model. Step 3 succinctly details the updating process for the polynomial coefficients of each base-learner. For the sake of completeness, we provide a detailed algorithm as follows:

Algorithm 6 Linear GeDS Boost fit polynomial coefficients update

Initialize: $i = 1$; $j = 1$; $\boldsymbol{\Delta}_{d_m,2} := \Delta_{d_{m-1},2}$; $\kappa = 1$
  while $i + j \leq d_{m-1} + 1 + \kappa_m + 1$ **do**
    $a_m^{(k)} = a_{m-1}^{(i)} + \nu \times {a^\dagger}_m^{(j)}$
    $b_m^{(k)} = b_{m-1}^{(i)} + \nu \times {b^\dagger}_m^{(j)}$
    $\kappa = \kappa + 1$;
    if $\delta_{j+2} < \xi_{i+2}$ **then**
      Add δ_{j+2} to $\boldsymbol{\Delta}_{d_m,2}$ between ξ_{i+1} and ξ_{i+2};
      $j \leftarrow j + 1$;
    else if $\delta_{j+2} > \xi_{i+2}$ **then**
      $i \leftarrow i + 1$;
    else
      $j \leftarrow j + 1$; $i \leftarrow i + 1$;
    end if
  end while

Affiliation:

Dimitrina S. Dimitrova, Vladimir K. Kaishev and Emilio L. Sáenz Guillén
Faculty of Actuarial Science and Insurance, Bayes Business School, City St George's, University of London, 106 Bunhill Row, EC1Y 8TZ London, United Kingdom
E-mail: D.Dimitrova@citystgeorges.ac.uk,
Vladimir.Kaishev.1@citystgeorges.ac.uk
Emilio.Saenz-Guillen@citystgeorges.ac.uk,
URL:
https://www.bayes.citystgeorges.ac.uk/faculties-and-research/experts/dimitrina-dimitrova,
https://www.bayes.citystgeorges.ac.uk/faculties-and-research/experts/vladimir-kaishev,
https://www.bayes.citystgeorges.ac.uk/faculties-and-research/students/emilio-saenz-guillen

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd