



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 been proposed as a flexible and efficient approach to spline regression by Kaishev, Dimitrova, Haberman, and Verrall (2016) and Dimitrova, Kaishev, Lattuada, and Verrall (2023). 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) into the GeDS framework, using GeD splines as base-learners to construct a boosted piecewise linear fit that serves as the control polygon for subsequent higher-order spline fits. 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. The numerical examples considered indicate that this technique can yield competitive spline fits, both in terms of accuracy and computational efficiency, relative to the boosting-with-splines procedure 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 in regression and classification settings, highlighting its potential for applications in broader data science and machine learning contexts.

Keywords: Variable-knot spline regression, gradient boosting, generalized additive models.

1. Introduction

Geometrically designed spline (GeDS) regression with variable knots provides a flexible alternative to existing methods in the free-knot regression splines literature of recent years (see [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 by 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](#); [Jupp 1978](#)), without relying on costly non-linear optimization. It yields competitive outcomes using 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 (GNMs)—that include generalized linear models (GLMs) 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, with favorable performance in the settings considered relative to other 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** ([Dimitrova, Kaishev, Lattuada, Sáenz Guillén, and Verrall 2026](#)), which is available from the Comprehensive R Archive Network (CRAN) at <https://CRAN.R-project.org/package=GeDS> and has attracted over 40,000 downloads since its release.

The R implementation of the canonical GeDS methodology can produce flexible fits using relatively few parameters (i.e., knots and regression coefficients), but is restricted to predictor dimensions 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 (GAMs; [Hastie and Tibshirani 1986, 1990](#)) provide a flexible statistical modeling technique that extends generalized linear models (GLMs) to allow for the modeling of predictor effects via non-linear *smooth functions* of the features. This makes it possible to capture more complex patterns while retaining the interpretability inherent in GLMs. Hastie and Tibshirani’s method to fit GAMs 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 2025](#)), which currently supports local regression and smoothing splines to fit the models’ smooth functions. This framework is extended by the R package **gamlss** ([Stasinopoulos and Rigby 2007](#)), which enables the modeling of all parameters of the conditional distribution of the response variable, rather than only the conditional mean. It is also extended by the R package **VGAM** ([Yee 2010, 2015](#)), which implements vector generalized additive models capable of concurrently

smoothing multiple linear (additive) predictors, thereby supporting multivariate responses and allowing the joint modeling of several distribution parameters for distributions beyond the exponential family. In contrast to the local-scoring/backfitting approach, Wood (2017) proposes a penalized regression spline approach with automatic smoothness selection. The latter approach is implemented in the **mgcv** R package (Wood 2025), which allows for more general smoothers than **gam**, including two-dimensional smoothers for spatial data or tensor product smoothers for interactions. Other popular software implementations of Hastie and Tibshirani’s GAM approach include PROC GAM in SAS (see Knafl and Ding 2016) and the community-contributed **gam** module in Stata (see Royston and Ambler 2002), while Wood’s approach is also implemented by **pyGAM** in Python (see Servén and Brummitt 2018).

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 argued 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 further extended 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, e.g., 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 current cumulative 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 in the **gamboostLSS** package (see Hofner, Mayr, and Schmid 2016) to allow for the simultaneous modeling of multiple distribution parameters, and in the **FDboost** package (see Brockhaus, Rügamer, and Greven 2020) by implementing boosting in functional regression settings with scalar and functional responses and covariates.

Trees are the most popular base procedure in boosting, and numerous widely recognized software implementations are available—see, e.g., the R package **gbm** (Ridgeway and GBM Developers 2024), as well as the more recent, highly optimized libraries, **XGBoost** (Chen and Guestrin 2016), **LightGBM** (Ke, Meng, Finley, Wang, Chen, Ma, Ye, and Liu 2017) or **CatBoost** (Dorogush, Ershov, and Gulin 2018). Nevertheless, splines have also been frequently considered. For instance, Bühlmann and Yu (2003) examined the L_2 Boost algorithm in detail and determined that L_2 Boost using smoothing splines as learners achieves optimal minimax rates of convergence in both regression and classification. The authors present L_2 Boosting 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_2 Boost 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_2 Boost, but with P-spline 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 2024).

In both GAM and boosting settings, the above-mentioned spline-focused approaches typically require the user to pre-specify the number of knots, degrees of freedom or related basis dimensions. 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 modeling framework within the exponential family of distributions. On the one hand, GAM-based extensions may be advantageous when the interpretability of models and visualization of effects are paramount. On the other hand, FGB-based extensions may be more suited to prediction-oriented settings, particularly when handling complex, high-dimensional data.

The numerical examples considered in this paper indicate that the above extensions can provide competitive spline fits, both in terms of accuracy and computational efficiency, in the selected settings. At the same time, they improve the flexibility and adaptability of the GeDS methodology, thereby broadening its applicability to more complex and diverse problem domains.

A key feature of FGB-GeDS is that the ability to express GeDS base-learners as piecewise polynomial functions enables the representation of the resulting 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 provides an automatic stopping rule, based on a ratio of consecutive deviances, for selecting the number of boosting iterations, offering an alternative to standard cross-validation-based procedures.

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 compare its behavior in selected numerical examples with approaches implemented in 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, we present generalized additive models with GeD splines. In Section 4, we outline the basic notions of functional gradient boosting, and in Section 5 the FGB-GeDS algorithm is introduced. In Sections 6, 7 and 8, we study the numerical properties of GAM-GeDS and FGB-GeDS and compare them with the estimators from the **gam** and **mgcv** packages, on the one hand, and those from the

mboost package, 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 describe the GeDS procedure for spline regression in terms of 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:

$$Y = f(X) + \epsilon \quad (1)$$

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 $S_{\mathbf{t}_{\kappa,n}}$ the linear space of all n -th order spline functions defined on a set of non-decreasing knots $\mathbf{t}_{\kappa,n} = \{t_i\}_{i=1}^{2n+\kappa}$, 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. $\mathbf{t}_{\kappa,n} = \{t_1 = \dots = t_n < t_{n+1} < \dots < t_{n+\kappa} < t_{n+\kappa+1} = \dots = t_{2n+\kappa}\}$. By the Curry-Schoenberg theorem, a spline regression function $f \in S_{\mathbf{t}_{\kappa,n}}$ can be expressed as

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

where $\boldsymbol{\theta} = (\theta_1, \dots, \theta_p)'$ is a vector of real-valued regression coefficients and $\mathbf{N}_n(x) = (N_{1,n}(x), \dots, N_{p,n}(x))'$ is the vector of $p = n + \kappa$ B-splines of order n defined on $\mathbf{t}_{\kappa,n}$. It is well known that $\sum_{i=j-n+1}^j N_{i,n}(t) = 1$ for any $t \in [t_j, t_{j+1})$, $j = n, \dots, n + \kappa$ 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 $\mathbf{t}_{\kappa,n}$ 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 stages. 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 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 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 [Kaishev *et al.* 2016](#), Section 3). The resulting LS linear spline fit is denoted by $\hat{f}(\boldsymbol{\delta}_{\kappa,2}; \hat{\boldsymbol{\alpha}}_p; x) = \sum_{i=1}^p \hat{\alpha}_i N_{i,2}(x)$ with number of internal knots κ , number of B-splines $p = \kappa + 2$ and knot locations $\boldsymbol{\delta}_{\kappa,2} = \{\delta_1 = \delta_2 < \delta_3 < \dots < \delta_{\kappa+2} < \delta_{\kappa+3} = \delta_{\kappa+4}\}$. The knot insertion stops when adding more knots does not significantly improve the fit according to the following residual sum of squares (RSS) criterion:

$$\text{RSS}(\kappa + q)/\text{RSS}(\kappa) = \sum_{i=1}^N \left(Y_i - \hat{f}(\boldsymbol{\delta}_{\kappa+q,2}; X_i) \right)^2 \bigg/ \sum_{i=1}^N \left(Y_i - \hat{f}(\boldsymbol{\delta}_{\kappa,2}; X_i) \right)^2 \geq \phi_{exit} \quad (3)$$

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 $q \geq 1$,

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

which is a generalization of (3), based on the deviance $D(\hat{\alpha}_{p+q}; \kappa + q; 2) = \sum_{i=1}^N \text{dev}(Y_i, \hat{\mu}_i)$, where $\hat{\mu}_i$ denotes the fitted mean for observation i under the corresponding spline model, and $\text{dev}(Y_i, \hat{\mu}_i)$ is its individual deviance contribution. 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 $\hat{f}(\delta_{\kappa+q,2}; \hat{\alpha}_{p+q}; x)$ does not significantly improve with the inclusion of the last q additional knots, and therefore, $\hat{f}(\delta_{\kappa,2}; \hat{\alpha}_p; x)$ 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 $\hat{f}(\delta_{\kappa,2}, \hat{\alpha}_p; x)$ from Stage A, with κ internal knots, the set of knots $\bar{t}_{\kappa-(n-2),n}$ (with $\kappa - (n - 2)$ internal knots) for each order $n = 3, \dots, n_{\max}$ is obtained by averaging the knots in the set $\delta_{\kappa,2}$ as follows:

$$\bar{t}_{i+n} = (\delta_{i+2} + \dots + \delta_{i+n}) / (n - 1), \quad i = 1, \dots, \kappa - (n - 2). \quad (5)$$

As shown by [Kaishev et al. \(2016\)](#), by choosing the knots $\bar{t}_{\kappa-(n-2),n}$ according to (5), the n -th order spline predictor curve $f(\bar{t}_{\kappa-(n-2),n}, \hat{\alpha}_p; x)$ becomes nearly the Schoenberg variation diminishing spline (VDS) approximation to the linear fit, $\hat{f}(\delta_{\kappa,2}, \hat{\alpha}_p; x)$, from Stage A. Consequently, the linear fit $\hat{f}(\delta_{\kappa,2}, \hat{\alpha}_p; x)$ can be viewed as (nearly) the control polygon of the predictor curve $f(\bar{t}_{\kappa-(n-2),n}, \hat{\alpha}_p; x)$. 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 $f(\bar{t}_{\kappa-(n-2),n}, \hat{\alpha}_p; x)$ obtained at Stage B.1 closely follows the shape of $\hat{f}(\delta_{\kappa,2}, \hat{\alpha}_p; x)$ and, thus, of the data. However, it is not formally a maximum likelihood fit to the data $\{Y_i, X_i\}_{i=1}^N$. To rectify this, Stage B.2 treats the B-spline coefficients $\hat{\alpha}_p$ as unknown parameters (now denoted by θ_p , $p = \kappa + 2$), and re-estimates them in a final run of the LS/IRLS procedure, while preserving the same set of knots $\bar{t}_{\kappa-(n-2),n}$.

Stage B.2

For each fixed order $n = 3, \dots, n_{\max}$, find the maximum likelihood estimates $\hat{\theta}_p$ of the B-spline coefficients, θ_p , of the spline predictor curve $f(\bar{t}_{\kappa-(n-2),n}, \theta_p; x)$ with knots determined in Stage B.1. Among all fits $\hat{f}(\bar{t}_{\kappa-(n-2),n}, \hat{\theta}_p; x)$, of order $n = 2, \dots, n_{\max}$ —i.e. including the linear fit, $\hat{f}(\delta_{\kappa,2}, \hat{\alpha}_p; x)$ 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 selects the spline order and, hence, its degree. 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.

In contrast to smoothing splines and P-splines, GeDS does not introduce an explicit roughness penalty in the fitting criterion. Regularization is instead induced internally by the residual-driven knot selection and stopping rule in Stage A, and by the variation-diminishing construction of smoother higher-order fits in Stage B. The Stage A stopping rule and the asymptotic properties of the GeDS estimator are studied in [Kaishev, Dimitrova, Haberman, and Verrall \(2006a\)](#), while the Stage B knot construction, its variation-diminishing optimal-

ity properties and error bounds are analyzed in [Kaishev, Dimitrova, Haberman, and Verrall \(2006b\)](#).

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, the `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 $Y \sim f(X)$ or $Z \sim f(X, Y)$. If needed, prior weights on observations can be assigned through the `weights` vector. Arguments `beta` and `phi` are numeric parameters in the intervals $[0, 1]$ and $(0, 1)$, respectively. 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.

Shape constraints can be imposed after the GeDS knot-selection step through the exported S3 generic `shapeConstrain()`. For a fixed GeDS-selected knot vector, monotonicity and convexity or concavity restrictions reduce to linear constraints on knot-scaled divided differences of the B-spline coefficients. The method applies to univariate Gaussian fits produced by `NGeDS()`, and to selected univariate smoothers or base-learners in Gaussian identity-link `NGeDSgam()` and FGB-GeDS `NGeDSboost()` fits (see Section 3 and Section 5, respectively). It acts on the fitted "GeDS", "GeDSgam" or "GeDSboost" object, retains the knots selected by the original fitting procedure and re-estimates the relevant B-spline coefficients under the chosen constraints, while holding the remaining additive components fixed.

3. Generalized additive models with GeDS

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

$$\mathbb{E}[Y|X^1, \dots, X^P] = \alpha + \sum_{j=1}^P f^j(X^j) \quad \text{or} \quad Y = \alpha + \sum_{j=1}^P f^j(X^j) + \varepsilon \quad (6)$$

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 $X^1, \dots, X^P$. 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 bivariate functions 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. Various techniques can be used to estimate additive models. In this context, the backfitting algorithm provides a flexible approach, since it allows the component functions to be updated iteratively using arbitrary scatterplot smoothers; see [Hastie and Tibshirani \(1990\)](#) and [Hastie et al. \(2009\)](#) for details.

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 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 $\mu = \mathbb{E}[Y|X^1, \dots, X^P]$. The model then relates μ to the predictor variables $X^1, \dots, X^P$ via a link function $g(\cdot)$:

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

Once a distribution for the response is specified, 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 $\mathbf{z}$ and the iteration-specific weights $\mathbf{w}$ (see Algorithm 1). Given these quantities, estimation of α and $f^1, \dots, f^P$ is

undertaken through the *local scoring algorithm* as detailed in Algorithm 1 (see [Hastie and Tibshirani 1990](#); [SAS Institute Inc. 2018](#)).

Algorithm 1 The General Local Scoring Algorithm

1. **Initialize:** $\hat{\alpha} = g\left(\frac{1}{N} \sum_{i=1}^N Y_i\right)$, $\hat{f}_m^j = 0$, $j = 1, \dots, P$, and $m = 0$.
 2. **Iterate:** Set $m = m + 1$ and iterate to form the predictor $\hat{\eta}$, the mean $\hat{\mu}$, the weights $\mathbf{w}$ and the adjusted dependent variable $\mathbf{z}$:
 - (a) Form the adjusted dependent variable
$$z_i = \hat{\eta}_i^{m-1} + (Y_i - \hat{\mu}_i^{m-1}) \cdot \left(\frac{\partial \hat{\eta}}{\partial \hat{\mu}}\right)_i^{m-1},$$
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 $i = 1, \dots, N$.
 - (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 $\mathbf{z}$ by using the backfitting algorithm (see Algorithm 4.1 in [Hastie and Tibshirani \(1990\)](#)) with weights $\mathbf{w}$ to obtain the estimated functions $\hat{f}_m^j(\cdot)$, $j = 1, \dots, P$, and the model $\hat{\eta}^m$.
 3. **Until:** The empirical deviance $D^m = \sum_{i=1}^N \text{dev}(Y_i, \hat{\mu}_i^m)$ fails to decrease.
-

Asymptotic properties of backfitting and local scoring estimators have been studied by [Buja, Hastie, and Tibshirani \(1989\)](#), [Opsomer \(2000\)](#), and [Kauermann and Opsomer \(2003\)](#); we refer to these works for details.

Our GAM-GeDS implementation involves applying the local scoring algorithm (Algorithm 1), using Normal GeD splines as the function smoothers within the backfitting algorithm. The method is implemented in the function `NGeDSgam()`, whose arguments are as follows:

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

The model is specified using a formula of the type $Y \sim f(X_1) + f(X_2) + \dots$, and the data should be provided as a `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 1, which is based on a ratio of deviances analogous to (4). Finally, `beta`, `phi`, `internal_knots` and `q` are used to tune the GeDS function smoothers, f^j , at each backfitting iteration.

4. Functional gradient boosting

In this section, we introduce the general framework of functional gradient boosting, which provides the foundation for the FGB-GeDS methodology presented in Section 5. We first describe the fundamental boosting algorithm and then briefly review its application to regression, classification and, more generally, responses from 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 random pair (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

$$F^*(\cdot) = \arg \min_{F(\cdot)} \mathbb{E} [L(Y, F(X))] \quad (8)$$

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 $L(\cdot, \cdot)$ over the joint distribution of (Y, X) (see Friedman *et al.* 2000). To enable gradient-based optimization of the loss, $L(\cdot, \cdot)$ is usually assumed to be differentiable and convex with respect to its second argument.

Given an i.i.d. learning sample of observations $(Y_1, X_1), \dots, (Y_N, X_N)$, estimation of $F^*(\cdot)$ is then undertaken by minimizing the empirical risk via iterative gradient descent in function space, that is, $\hat{F}(\cdot) = \arg \min_{F(\cdot)} \frac{1}{N} \sum_{i=1}^N L(Y_i, F(X_i))$. Functional gradient boosting, as given by Friedman (2001), is presented in Algorithm 2.

Algorithm 2 Functional Gradient Boosting

Given data $\{Y_i, X_i\}_{i=1}^N$, a differentiable loss function $L(Y, F(X))$ 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: $\hat{F}_0(\cdot) = \arg \min_c \frac{1}{N} \sum_{i=1}^N L(Y_i, c)$, where c is a constant that minimizes the average loss.
 2. For $m = 1$ to m_{stop} ,
 - i Compute the negative gradient vector:

$$U_{i,m} = - \left[\frac{\partial L(Y_i, F(X_i))}{\partial F(X_i)} \right]_{F(X_i) = \hat{F}_{m-1}(X_i)} \quad i = 1, \dots, N$$
 - ii Fit a real-valued base (weak) learner $\hat{f}_m$ to the gradient vector $U_{i,m}$.
 - iii Find the step size parameter ν_m as

$$\nu_m = \arg \min_{\nu} \sum_{i=1}^N L(Y_i, \hat{F}_{m-1}(X_i) + \nu \hat{f}_m(X_i))$$
 - iv Update the model: $\hat{F}_m(\cdot) = \hat{F}_{m-1}(\cdot) + \nu_m \hat{f}_m(\cdot)$
-

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 “early 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 size of each boosting update must be determined. According to [Friedman \(2001\)](#), an iteration-specific step length ν_m should be obtained by minimizing the objective (loss) function at each boosting iteration (see Step 2.iii in [Algorithm 2](#)). This methodological choice explains why Friedman’s approach is often termed “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 a fixed step-length or shrinkage parameter 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 values of m_{stop} , and vice versa (see [Friedman 2001](#)).

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

$$\hat{F}_{m_{\text{stop}}}(\cdot) = \hat{F}_0(\cdot) + \sum_{l=1}^{m_{\text{stop}}} \nu_l \hat{f}_l(\cdot), \quad (9)$$

that is, as a sum of base procedure estimates. Consequently, 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 introduce a novel boosting algorithm that uses GeD splines as the base procedure (see [Section 5](#)). Before doing so, we briefly review the application of boosting.

4.1. Boosting for regression

The choice of loss function determines the negative gradient fitted at each boosting iteration. 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:

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

That is, the previously mentioned L_2 Boost algorithm implements the general FGB algorithm presented in [Algorithm 2](#) using (10) as the loss function. The corresponding population minimizer of the L_2 -loss is

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

and thus L_2 Boost 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_2 Boost 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_2 Boost benefits from a straightforward iterative structure given that the negative gradient of the squared-error loss is simply the residual vector. As a result, the algorithm boils down to re-fitting the residuals from the previous boosting iteration at each new iteration.

Other losses may be preferable in less standard regression settings. For example, the L_1 -loss targets the conditional median and is more robust to large residuals, while the Huber loss

provides a compromise between L_2 and L_1 losses in the presence of moderately heavy-tailed errors (see, e.g., [Hastie et al. 2009](#); [Karunasingha 2022](#)).

4.2. Boosting for classification and exponential-family responses

Boosting can be effectively applied in classification scenarios as well. Given a binary outcome variable $Y \in \{0, 1\}$ with $Y|X = x \sim \text{Bernoulli}(p(x))$, it is usual to model the posterior probability $p(x) = P(Y = 1|X = x)$ of an instance belonging to the positive class through the log-odds (logit) function.

Although the misclassification loss is the natural target for classification, it is discontinuous and non-convex, and is therefore not suitable for gradient-based optimization. For this reason, functional gradient boosting typically optimizes a convex differentiable surrogate, such as the negative log-likelihood of the binomial model (see [Friedman 2001](#); [Bühlmann and van de Geer 2011](#)). In particular, after writing $\tilde{y} = 2y - 1$ for notational convenience and rescaling, this loss function can be written as

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

which is convex and differentiable in F . The corresponding population minimizer is

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

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 $Y \in \{0, 1, 2, \dots\}$, we consider Poisson regression, that is, assume $Y|X = x \sim \text{Poisson}(\lambda(x))$, and implement FGB using the following loss function:

$$L(y, F(x)) = -yF(x) + \exp(F(x)), \quad \text{where} \quad F^*(x) = \log(\lambda(x)). \quad (12)$$

4.3. Asymptotic properties of boosting and software implementations

The asymptotic properties of boosting estimators have been studied extensively. For L_2 Boost and related spline-based boosting procedures, see [Bühlmann and Yu \(2003\)](#), [Bühlmann and Hothorn \(2007\)](#), [Yao, Rosasco, and Caponnetto \(2007\)](#) and [Bissantz, Hohage, Munk, and Ruymgaart \(2007\)](#). Consistency results for high-dimensional and component-wise boosting settings are discussed, among others, by [Bühlmann \(2006\)](#) and [Yousuf and Ng \(2021\)](#). Broader reviews and additional references are given in [Bühlmann and Hothorn \(2007\)](#) and [Liang and Sur \(2022\)](#).

Functional gradient boosting has been extensively implemented across various statistical software ecosystems. In R, a prominent and comprehensive implementation is provided by the **mboost** package ([Hothorn et al. 2024](#)), which supports flexible base-learner specification, including trees, linear models and P-splines. This model-based boosting framework has been extended by the **gamboostLSS** and **FDboost** packages (see [Hofner et al. 2016](#); [Brockhaus et al. 2020](#), respectively).

Other widely used R packages include **gbm** (Ridgeway and GBM Developers 2024), which closely follows Friedman’s original algorithm using regression trees, as well as the high-performance libraries **XGBoost**, **LightGBM** and **CatBoost**, which are also implemented in Python. For broader machine learning workflows, the package **caret** (Kuhn 2008), its successor **tidymodels** (Kuhn and Wickham 2020) or the **mlr/mlr3** packages (Bischl, Lang, Kotthoff, Schiffner, Richter, Studerus, Casalicchio, and Jones 2016; Lang, Binder, Richter, Schratz, Pfisterer, Coors, Au, Casalicchio, Kotthoff, and Bischl 2019) provide comprehensive and widely adopted toolkits in R, as does the **scikit-learn** library (Pedregosa, Varoquaux, Gramfort, Michel, Thirion, Grisel, Blondel, Prettenhofer, Weiss, Dubourg, VanderPlas, Passos, Cournapeau, Brucher, Perrot, and Duchesnay 2011) in Python; all support a variety of gradient boosting algorithms. In contrast to model-based boosting approaches such as **mboost**, which yield interpretable additive model structures, these tree-based methods primarily aim at optimizing predictive performance (see, e.g., Brockhaus *et al.* 2020). Because FGB-GeDS is itself a model-based boosting approach, we restrict our empirical comparison to the **mboost** implementation.

5. L_2 Boost 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 3). 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 to re-express 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 direct consolidation into an explicit piecewise-polynomial model is not generally provided by standard boosting implementations using, e.g., smoothing splines as base-learners (cf. Schmid and Hothorn 2008a), for which 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 vector of residuals—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 3 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):

$$\hat{F}_0(\Delta_{d_0,2}; \cdot) = \hat{f}_0(\delta_{\kappa_0,2}, \hat{\alpha}_{p_0}; \cdot)$$

where $\delta_{\kappa_0,2}$ denotes the set of knots of the linear base-learner *GeDS fit* (i.e., of order $n = 2$) and $\hat{\alpha}_{p_0}$ is the respective vector of $p_0 = \kappa_0 + 2$ estimated B-spline coefficients. We will denote by $\Delta_{d_m,2} = \{\xi_1 = \xi_2 < \xi_3 < \dots < \xi_{d_m+3} = \xi_{d_m+4}\}$ 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 $\{\delta_{\kappa_0,2} \cup \dots \cup \delta_{\kappa_m,2}\}$.

Convert the linear GeDS fit—which has a B-spline representation—into piecewise polynomial form, defined by the distinct knots $\{\delta_2 < \delta_3 < \dots < \delta_{\kappa_0+2} < \delta_{\kappa_0+3}\} \in \delta_{\kappa_0,2}$ 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*, $\hat{f}_0(\delta_{\kappa_0,2}, \hat{\alpha}_{p_0}; \cdot)$ —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, $\hat{F}_m(\Delta_{d_m,2}; \cdot)$, which are updated at each boosting iteration.

1.b. Compute the negative gradient vector of the loss function and evaluate it at $\hat{F}_m(\Delta_{d_m,2}; X_i)$. For the L_2 loss, this amounts to computing the residual vector:

$$U_{i,m+1} = - \left[\frac{\partial L(Y_i, F(X_i))}{\partial F(X_i)} \right]_{F(X_i) = \hat{F}_m(\Delta_{d_m,2}; X_i)} = Y_i - \hat{F}_m(\Delta_{d_m,2}; X_i) = r_{i,m+1}, \quad (13)$$

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

¹For simplicity, we present the algorithm in terms of univariate 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, $\delta_{\kappa_m,2}$, 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, $\{\delta_2 < \dots < \delta_{\kappa_m+3}\}$, from $\delta_{\kappa_m,2}$, and identify the corresponding polynomial coefficients. We denote the intervals defined by these breakpoints as $\mathcal{I}^{GeDS} = \{\mathcal{I}_j^{GeDS} = [\delta_{j+1}, \delta_{j+2}), j = 1, \dots, \kappa_m + 1\}$.

Step 3. Let $\mathcal{I}^{m-1} = \{\mathcal{I}_i^{m-1} = [\xi_{i+1}, \xi_{i+2}), i = 1, \dots, d_{m-1} + 1\}$, $d_{m-1} + 1 \leq \sum_{l=0}^{m-1} (\kappa_l + 1)$, 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

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

where $0 < \nu \leq 1$ is a real-valued step length/shrinkage factor². 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:

$$\hat{F}_m(\Delta_{d_m,2}; \cdot) = \hat{F}_{m-1}(\Delta_{d_{m-1},2}; \cdot) + \nu \times \hat{f}_m(\delta_{\kappa_m,2}, \hat{\alpha}_{p_m}; \cdot)$$

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

Step 4. Recompute the residuals, as specified in (13), with respect to the model $\hat{F}_m(\Delta_{d_m,2}; \cdot)$. Let $q_{\text{boost}} \geq 1$ be some predefined integer. If $m < q_{\text{boost}}$, go back to Step 2; otherwise calculate the ratio:

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

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

Step 5. Repeat Steps 2–4 until $\phi_{\text{boost}} \geq \phi_{\text{boost}}^{\text{exit}}$, where $\phi_{\text{boost}}^{\text{exit}} \in (0, 1)$ is chosen close to 1. Note that, under a general loss function, the empirical risk is not guaranteed to decrease at each boosting iteration. 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. It may 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 $q_{\text{boost}} + 1$ models, minimizes the empirical residual sum of squares:

$$\hat{F}^*(\Delta_{d_m^*,2}; \cdot) = \arg \min_{\hat{F}_l(\Delta_{d_l,2}; \cdot)} \sum_{i=1}^N \left(Y_i - \hat{F}_l(\Delta_{d_l,2}; X_i) \right)^2, \quad l = m - q_{\text{boost}}, \dots, m.$$

²As 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 $\hat{F}^*(\Delta_{d_m^*,2}; \cdot)$ obtained in Step 6. This has d_m^* internal knots, with locations given by $\Delta_{d_m^*,2}$. We compute the higher-order fits, specifically for $n = 3$ and $n = 4$, by first calculating the internal knot placement of the higher-order knot vector $\bar{\tau}_{*,n}$ ³. This is defined through:

$$\tau_{i+n} = (\Delta_{i+2} + \dots + \Delta_{i+n}) / (n - 1), \quad i = 1, \dots, d_m^* - (n - 2) \quad (14)$$

Second, we find the least-squares fit $\hat{F}^*(\bar{\tau}_{*,n}, \hat{\theta}; \cdot)$ that solves

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

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 $\phi_{\text{boost}} \geq \phi_{\text{boost}}^{\text{exit}}$, the performance of $\hat{F}_m(\Delta_{d_m,2}; \cdot)$ has not significantly improved (and may even have worsened) over the preceding q_{boost} iterations. Consequently, the iterations are stopped, and $\hat{F}^*(\Delta_{d_m^*,2}; \cdot)$ 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 $\Delta_{d_m^*,2}$, transformed according to the *averaging knot location* method described by [Kaishev et al. \(2016\)](#) and given in (14). 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., a maximum of two internal knots) and then perform a few boosting iterations using GeDS learners operating at their full potential (i.e., without imposing a restrictive maximum number of knots); that is, pre-define the maximum number of internal knots for the initial learner, κ_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 $\hat{f}_m(\delta_{\kappa_m,2}, \hat{\alpha}_{p_m}; \cdot)$ —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 corresponding to models separated by q_{boost} consecutive boosting iterations (cf. Step 4 in Algorithm 3). 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 scale the contribution 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).

³The choice of the knots $\bar{\tau}_{*,n}$, according to [Kaishev et al. \(2016\)](#), ensures that the n th-order spline predictor curve $\hat{F}^*(\bar{\tau}_{*,n}; \cdot)$ becomes nearly the VDS approximation to $\hat{F}^*(\Delta_{d_m^*,2}; \cdot)$.

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 2024). 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 $(Y_1, \dots, Y_N)^\top$ is the response vector and $X_1, \dots, X_N$ 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:

$$\hat{F}^* = \hat{F}_0 + \hat{F}_1^* + \dots + \hat{F}_K^* \quad \text{with} \quad \hat{F}_j^* = \nu \times \sum_{l=1}^{m^*} \hat{f}_l^j \mathbb{1}(\hat{\mathbf{f}}_l = \hat{f}_l^j), \quad j = 1, \dots, K \quad (15)$$

where m^* stands for the boosting iteration where the final fit $\hat{F}^*$ is achieved; $\hat{\mathbf{f}}_l$ denotes the base-learner selected at the l th boosting iteration, according to some pre-established criterion. Each function estimate $\hat{F}_j^*$ is then the cumulative sum of the estimates $\hat{f}_l^j$ for each iteration where the corresponding j^{th} base-learner was selected to update the model $\hat{F}$, i.e., for each iteration where $\hat{\mathbf{f}}_l = \hat{f}_l^j$. Algorithm 4 describes our implementation of component-wise gradient boosting using the L_2 loss function and GeD splines as base-learners.

Algorithm 4 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 GeDS base-learners, which may generally be univariate or bivariate. For simplicity, we present the algorithm for $K = P$ univariate base-learners, with one base-learner for each predictor variable. Separately fit K linear GeDS models $Y \sim \hat{f}_0^j(X^j)$ with κ_0^j internal knots each, $j = 1, \dots, K$. Set $\hat{F}_0$ to be the base-learner fit that minimizes the sum of squared residuals, that is:

$$\hat{F}_0(\Delta_{d_0,2}; \cdot) = \arg \min_{\substack{\hat{f}_0^j \\ j=1, \dots, K}} \sum_{i=1}^N \left(Y_i - \hat{f}_0^j(\delta_{\kappa_0^j, 2}; \hat{\alpha}_{p_0^j}; X_i^j) \right)^2$$

1.b. Initialize $m = 0$. Compute the negative gradient vector of the L_2 loss function:

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

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

$$\hat{\mathbf{f}}_m = \arg \min_{\substack{\hat{\mathbf{f}}_m^j \\ j=1, \dots, K}} \sum_{i=1}^N (r_{i,m} - \hat{\mathbf{f}}_m^j(\boldsymbol{\delta}_{\kappa_m^j, 2}^j; \hat{\boldsymbol{\alpha}}_{p_m^j}^j; X_i^j))^2.$$

Step 3. Update the current estimate as

$$\hat{F}_m(\cdot) = \hat{F}_{m-1}(\cdot) + \nu \times \hat{\mathbf{f}}_m$$

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

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

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

Step 5. Repeat Steps 2–4 until $\phi_{\text{boost}} \geq \phi_{\text{boost}}^{\text{exit}}$, where $\phi_{\text{boost}}^{\text{exit}} \in (0, 1)$ 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:

$$\hat{F}^*(\cdot) = \arg \min_{\hat{F}_l(\cdot)} \sum_{i=1}^N \left(Y_i - \hat{F}_l(\cdot; X_i) \right)^2, \quad l = m - q_{\text{boost}}, \dots, m.$$

Step 7. Step 6 yields a final linear fit, $\hat{F}^*(\cdot)$, formed only by the base-learners selected up to iteration m^* . Let

$$\mathcal{J}^* = \left\{ j \in \{1, \dots, K\} : m_j^* > 0 \right\} \text{ and } m_j^* = \sum_{m=0}^{m^*} \mathbb{1}(\hat{\mathbf{f}}_m = \hat{\mathbf{f}}_m^j),$$

denote the set of selected base-learners and the number of times each of them is selected. For each $j \in \mathcal{J}^*$, the final linear fit has $d_{m^*}^j, d_{m^*}^j \leq \sum_{l=0}^{m^*} \kappa_l^j \mathbb{1}(\hat{\mathbf{f}}_l = \hat{\mathbf{f}}_l^j)$, internal knots with knot locations $\Delta_{d_{m^*}^j, 2}^j$. We compute the higher-order fits (quadratic and cubic) by first calculating, for each selected base-learner, the knot placement $\bar{\tau}_{*,n}^j$, defined as

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

We then find the least-squares fit $\hat{F}^*(\bar{\tau}_{*,n}, \hat{\boldsymbol{\theta}}; \cdot)$ that solves

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

The function `NGeDSboost()` in the **GeDS** package implements functional gradient boosting with GeD splines, following Algorithm 3, 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 4. 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, q = 2L,
higher_order = TRUE, boosting_with_memory = FALSE)
```

The model is specified using a formula of the type $Y \sim f(X_1) + f(X_2) + \dots$ 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`. A maximum and minimum number of boosting iterations can be set through `max_iterations` and `min_iterations`, respectively, although in practice the number of boosting iterations is primarily regulated by the stopping rule, which is tuned by `phi_boost_exit` and `q_boost` (see Step 4 in Algorithms 3 and 4). 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 $\kappa_0^{\max}$ is set via `int.knots_init`, and the maximum number of internal knots of the base-learner(s) at each boosting iteration $\kappa_m^{\max}$ can be set via `int.knots_boost`. In practice, however, we recommend leaving `int.knots_boost` sufficiently large, allowing the complexity of the GeDS base-learners to be regulated by the GeDS stopping rule itself.

The proposed FGB-GeDS implementation differs from a standard boosting workflow in which a fixed base-learner is supplied to a generic boosting engine. First, the procedure can start from a flexible GeDS initial learner with a small prespecified maximum number of internal knots, rather than only from the empirical risk minimizer. Second, at each boosting iteration, the complexity of the GeDS learner is adaptively determined through the GeDS knot-selection procedure, based on the structure of the current negative gradient; in many settings, this reduces the need to rely on small shrinkage parameters to control each update. Third, the number of boosting iterations is selected internally through the deviance-ratio stopping rule (see Step 4 in Algorithms 3 and 4), rather than being fixed in advance or selected *a posteriori* by cross-validation. Fourth, each boosting iteration produces a single piecewise-polynomial spline fit, so that the final FGB-GeDS model is also represented as a single explicit spline rather than as an additive expansion of base-learner updates. Finally, the GeDS learners fitted during boosting are linear splines that are successively combined into a single final control polygon. The final higher-order fits are then constructed from this control polygon, preserving the geometric design principle of the GeDS methodology: first constructing a data-adaptive control polygon that captures the shape of the underlying signal, and then deriving smoother higher-order spline fits that closely follow it.

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 3. 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.

A wide range of other spline-based alternatives have been proposed for estimating the linear predictor in generalized linear and nonlinear models, and have already been thoroughly compared to GeDS in the univariate (non-additive) case by [Dimitrova et al. \(2023\)](#). These include Semi-parametric Models (SPM) by [Eilers and Marx \(1996\)](#) available in **SemiPar** ([Wand 2018](#)); Generalized Smoothing Spline (GSS) ANOVA models introduced by [Wahba, Wang, Gu, Klein, and Klein \(1995\)](#) and implemented in **gss** ([Gu 2014](#)); and Generalized Additive Models ([Wood 2017](#)), implemented in **mgcv** ([Wood 2025](#)) and considered in the additive case in Section 7.

Beyond R, generic spline regression functionality is also available in other programming environments. In Python, the **statsmodels** package ([Seabold and Perktold 2010](#)), in combination with **patsy** ([Smith, Wardrop, and Capretto 2024](#)) or its modern, high-performance alternative **formulaic** ([Wardrop 2024](#)), provides an implementation of spline regression, allowing for the inclusion of spline terms in linear models, generalized linear models and generalized additive models (currently in “experimental status”), via `bs()` for B-splines, `cr()` for natural cubic splines and `cc()` for cyclic cubic splines; **patsy** additionally provides `te()` for tensor-product splines. In Stata ([StataCorp 2025](#)), regression splines are implemented via the `makespline()` command, which supports B-splines (`bspline()`), piecewise polynomial splines (`piecewise()`), restricted (natural) cubic splines (`racs()`) and linear splines (`linear()`) for use in standard regression models. However, these implementations are not included in the numerical comparisons because their spline bases use knots placed at predetermined locations, typically based on empirical quantiles or uniform spacing. This general fixed-knot approach is already represented in our comparisons by the P-spline base-learners implemented in **mboost**.

6.1. FGB-GeDS fitting process

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

$$f_1(x) = 40 \frac{x}{1 + 100x^2} + 4, \quad x \in [-2, 2] \quad (16)$$

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., $Y_i \sim \mathcal{N}(\mu_i, \sigma^2)$ 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 $X_i \sim U[-2, 2]$, $i = 1, \dots, N$, 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, $\Delta_{d_m, 2}$, 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 (14), 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 3. 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:n.boost.iter(Gmodboost), final_fits = TRUE)

```

6.2. Further examples and model comparison

Bühlmann and Yu (2003) empirically demonstrate the competitiveness of L_2 Boosting with component-wise 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

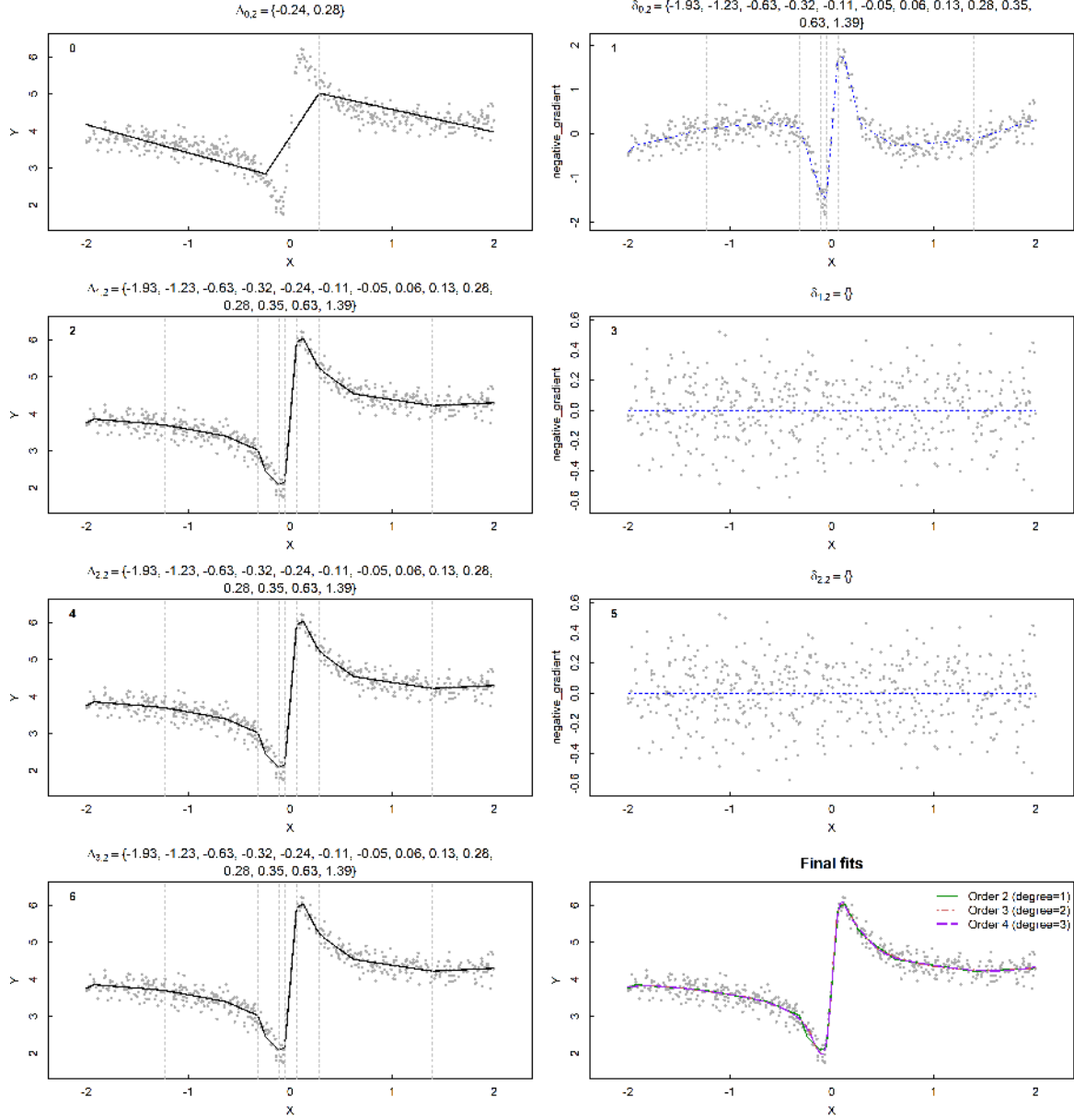


Figure 1: (Left) Linear `NGeDSboost()` fit over the data at each boosting iteration. (Right) `NGeDS()` fit (i.e., base-learner fit) to the recomputed residuals at each boosting iteration. The last plot depicts the final linear fit together with the higher-order fits (quadratic and cubic).

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](#); [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 vary depending on the number of folds and the method of partitioning the data, potentially affecting the choice of m_{stop} . Overall, determining m_{stop} via cross-validation inherently constitutes an *ex-post* approach, since the stopping iteration is chosen after evaluating a fitted boosting path. This may be less natural in applications where data are collected sequentially, and the model is expected to stop according to information available during the fitting process. In this context, the stopping rule presented in Step 4 of Algorithm 3 provides an automatic alternative based on the ratio of consecutive deviances. 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 cases.

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. We also add two stress-test scenarios: one based on a heavy-tailed version of the Doppler setting in Example 6.2, in which the Gaussian errors are replaced by rescaled Student-t errors, and one based on the Blocks function, introduced as Example 6.3 in Appendix B. Finally, we consider a real data application in Example 6.4.

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

$$f_2(x) = 5\sqrt{x(1-x)} \sin \frac{2\pi(1+0.05)}{(x+0.05)}, \quad x \in [0, 1] \quad (17)$$

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 $Y_i \sim \mathcal{N}(\mu_i = f_2(X_i), \sigma^2)$, $\sigma = 0.2$, or $Y_i \sim t_3(\mu_i = f_2(X_i), s = 0.2/\sqrt{3})$, where s denotes the scale parameter, and $X_i = (i-1)/N$, $i = 1, \dots, N$, $N = 400$.

```
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)
```

⁴Note 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”.

```

R> # Add (Normal) noise to the mean of Y
R> Y <- rnorm(N, means, sd = 0.2)
R> # Add heavy-tailed noise to the mean of Y
R> # The t errors are rescaled so that their distribution has the same
R> # standard deviation as the baseline Normal errors, sd = 0.2
R> nu <- 3
R> eps <- rt(N, df = nu)
R> eps <- eps * 0.2 / sqrt(nu / (nu - 2))
R> Y <- means + eps

```

We now compare the performance of `NGeDS()`/`GGeDS()`, `NGeDSboost()` and `mboost()` for examples 6.1, 6.2 and 6.3. Before presenting the results, we make explicit which tuning choices correspond to package defaults and which were set specifically for these examples.

For `NGeDS()`/`GGeDS()` and `NGeDSboost()`, the GeDS stopping-rule parameters are generally left at their default values, namely `phi = 0.99` and `q = 2`. The exception is the Poisson `NGeDSboost()` fit in Example 6.1, for which `phi = 0.9`. In Example 6.1, the knot-placement parameter β is left at its default value for each method and response family: `beta = 0.5` for `NGeDS()` and `NGeDSboost()`, `beta = 0.2` for Poisson `GGeDS()` and `beta = 0.1` for Gamma `GGeDS()`. It is set to `beta = 0.6` in the Doppler and Blocks settings, introduced in Examples 6.2 and 6.3, respectively. The values `beta = 0.5` and `beta = 0.6` are within the range recommended by Kaishev *et al.* (2016) for “wiggly underlying functions and high signal-to-noise ratio data”. 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 (cf. Kaishev *et al.* 2016).

For `NGeDSboost()`, the boosting stopping-rule parameters are also kept at their defaults, namely, `phi_boost_exit = 0.99` and `q_boost = 2`. The initial learner parameters are likewise left at their defaults, `initial_learner = TRUE` and `int.knots_init = 2`. The shrinkage rate is also kept at its default value, `shrinkage = 1`, except in the Poisson and Gamma versions of Example 6.1, where it is set to `shrinkage = 0.005` and `shrinkage = 0.1`, respectively. These values are chosen to account for the sensitivity of the logarithmic link function associated with these distributions. The lower shrinkage rates prevent overly large updates resulting from the exponential scale of the response for these distributions, hence ensuring stable updates at each boosting iteration.

The function `mboost()` is run using a smooth P-spline base-learner, with the spline degree left at its default value of three (cubic). 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 corresponding median performance measure, namely MSE for the Gaussian examples and mean deviance for the Poisson and Gamma examples relative to the corresponding true functions, based on the fits to 10 simulated data samples. For the Blocks example, the degree of the P-spline was also included in the tuning grid, since the best-performing GeDS-based fits in this setting were linear. 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 of knots selected by `NGeDSboost()` across the simulations in each example. Following the recommendations in the `mboost` package documentation, we set the shrinkage rate to its

default value, 0.1. For the Poisson example, however, we use a smaller shrinkage rate, 0.005, as we did in `NGeDSboost()`, which gave better performance in this setting. We also increase m_{stop} to 10,000 boosting iterations, after observing that this choice led to better outcomes in these examples than both the default setting $m_{\text{stop}} = 100$ and the intermediate choice $m_{\text{stop}} = 1,000$.

The models are tested for examples 6.1, 6.2 and 6.3, by fitting 100 different simulated datasets. 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, in 6.2 and in 6.3, fits are compared according to their MSE with respect to the true generating function, defined as $\left\{ \sum_{i=1}^N \left(f(X_i) - \hat{f}(X_i) \right)^2 \right\} / N$. 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.

Table 1 displays the median performance measure for each model, namely MSE for the Gaussian examples and mean deviance for the non-Gaussian examples, with the first and third quartiles in square brackets, together with the median number of boosting iterations, median number of internal knots and median fitting time.

Both `NGeDS()`/`GGeDS()` and `NGeDSboost()` yield low median MSEs and median mean deviances across the considered examples. `NGeDSboost()` improves slightly on the best corresponding `NGeDS()`/`GGeDS()` fits in Example 6.1, while the two GeDS-based approaches give very similar median errors in the Doppler and Blocks settings.

For the first `mboost()` implementation considered, in which both the number of knots and the degrees of freedom of the P-spline are pre-tuned, Table 1 shows that the median MSEs/median mean deviances remain above those of the best-performing GeDS-based fits in most settings, with the exception of the Blocks example. In addition, note that the GeDS-based fits use fewer internal knots, and `NGeDSboost()` requires fewer boosting iterations than `mboost()`. When the number of knots in `mboost()` is set equal to the median number of knots of the final FGB-GeDS fits, and only the degrees of freedom are tuned, the corresponding MSEs/mean deviances increase noticeably.

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, to ensure consistency across the three response distributions. As it can be observed, `NGeDSboost()` indeed boosts the performance of the already competitive canonical GeDS for example 6.1, with slightly lower median errors and tighter variability in the cases considered. When both the number of knots and d.f. are tuned, `mboost()` gives comparable results in the Normal and Poisson cases, although its MSEs/mean deviances are generally higher than those of the cubic FGB-GeDS fits. In the Gamma case, the pre-tuned `mboost()` specification shows more variable results, as reflected by the wider interquartile range and longer whiskers in the

	Type	MSE/Mean Dev.	Boosting iter.	Internal knots	Time (sec.)
Example 6.1					
<i>Normal</i>					
NGeDS()	Linear	0.0028 [0.0022, 0.0039]		10	0.01
	Quadratic	0.0017 [0.0013, 0.0024]			
	Cubic	0.0016 [0.0012, 0.0025]			
NGeDSboost()	Linear	0.0026 [0.0021, 0.0032]	3	12	0.03
	Quadratic	0.0014 [0.0011, 0.0019]			
	Cubic	0.0013 [0.0010, 0.0017]			
mboost()	Tuned d.f. & knots	0.0032 [0.0029, 0.0038]	10,000	42	1.60
	Tuned d.f.	0.0524 [0.0477, 0.0568]	10,000	12	1.09
<i>Poisson</i>					
NGeDS()	Linear	0.0938 [0.0757, 0.1285]		14	0.04
	Quadratic	0.0525 [0.0434, 0.0930]			
	Cubic	0.0506 [0.0378, 0.0784]			
NGeDSboost()	Linear	2.5530 [2.3145, 2.9300]	4	18	0.05
	Quadratic	0.0540 [0.0433, 0.0660]			
	Cubic	0.0482 [0.0385, 0.0585]			
mboost()	Tuned d.f. & knots	0.0919 [0.0790, 0.1057]	10,000	42	2.82
	Tuned d.f.	1.4178 [1.2470, 1.5959]	10,000	18	2.22
<i>Gamma</i>					
NGeDS()	Linear	0.0024 [0.0018, 0.0029]		13	0.06
	Quadratic	0.0016 [0.0013, 0.0023]			
	Cubic	0.0035 [0.0020, 0.0057]			
NGeDSboost()	Linear	0.0661 [0.0417, 0.0866]	2	16	0.06
	Quadratic	0.0013 [0.0010, 0.0017]			
	Cubic	0.0017 [0.0014, 0.0021]			
mboost()	Tuned d.f. & knots	0.0314 [0.0178, 0.0491]	10,000	81	6.80
	Tuned d.f.	0.0272 [0.0252, 0.0294]	10,000	16	7.57
Example 6.2					
<i>Baseline Doppler</i>					
NGeDS()	Linear	0.0220 [0.0172, 0.0367]		36	0.08
	Quadratic	0.0217 [0.0161, 0.0452]			
	Cubic	0.0375 [0.0227, 0.0681]			
NGeDSboost()	Linear	0.0223 [0.0167, 0.0369]	3	39	0.10
	Quadratic	0.0227 [0.0163, 0.0495]			
	Cubic	0.0408 [0.0236, 0.0672]			
mboost()	Tuned d.f. & knots	0.0540 [0.0532, 0.0550]	10,000	92	1.89
	Tuned d.f.	0.1478 [0.1473, 0.1485]	10,000	39	3.04
<i>Heavy-tailed Doppler</i>					
NGeDS()	Linear	0.0279 [0.0184, 0.0475]		35	0.05
	Quadratic	0.0287 [0.0178, 0.0655]			
	Cubic	0.0436 [0.0264, 0.1033]			
NGeDSboost()	Linear	0.0287 [0.0200, 0.0456]	3	37	0.06
	Quadratic	0.0304 [0.0196, 0.0617]			
	Cubic	0.0432 [0.0291, 0.0904]			
mboost()	Tuned d.f. & knots	0.0530 [0.0521, 0.0545]	10,000	92	1.29
	Tuned d.f.	0.2055 [0.2048, 0.2063]	10,000	37	1.56
Example 6.3					
<i>Blocks</i>					
NGeDS()	Linear	0.0082 [0.0042, 0.0357]		81	1.46
	Quadratic	0.0130 [0.0078, 0.0472]			
	Cubic	0.0263 [0.0194, 0.0810]			
NGeDSboost()	Linear	0.0093 [0.0049, 0.0391]	3	83	0.49
	Quadratic	0.0120 [0.0078, 0.0491]			
	Cubic	0.0253 [0.0190, 0.0814]			
mboost()	Tuned d.f., knots & degree	0.0038 [0.0036, 0.0041]	10,000	199	2.49
	Tuned d.f. & degree	0.1899 [0.1898, 0.1901]	10,000	83	2.78

Table 1: Median mean squared error (MSE)/median mean deviance, with first and third quartiles reported in square brackets, together with the median number of boosting iterations, median number of internal knots and median computation time for NGeDS(), NGeDSboost() and mboost() over 100 simulated datasets for Examples 6.1, 6.2 and 6.3.

boxplot. Across all cases of example 6.1, the MSEs/mean deviances for mboost() are higher when the number of internal 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 behavior of the Doppler function near the origin when the number of knots is fixed to the median used by `NGeDSboost()` and only the d.f. are tuned. In contrast, the GeDS models provide a fairly good fit across the entire function range.

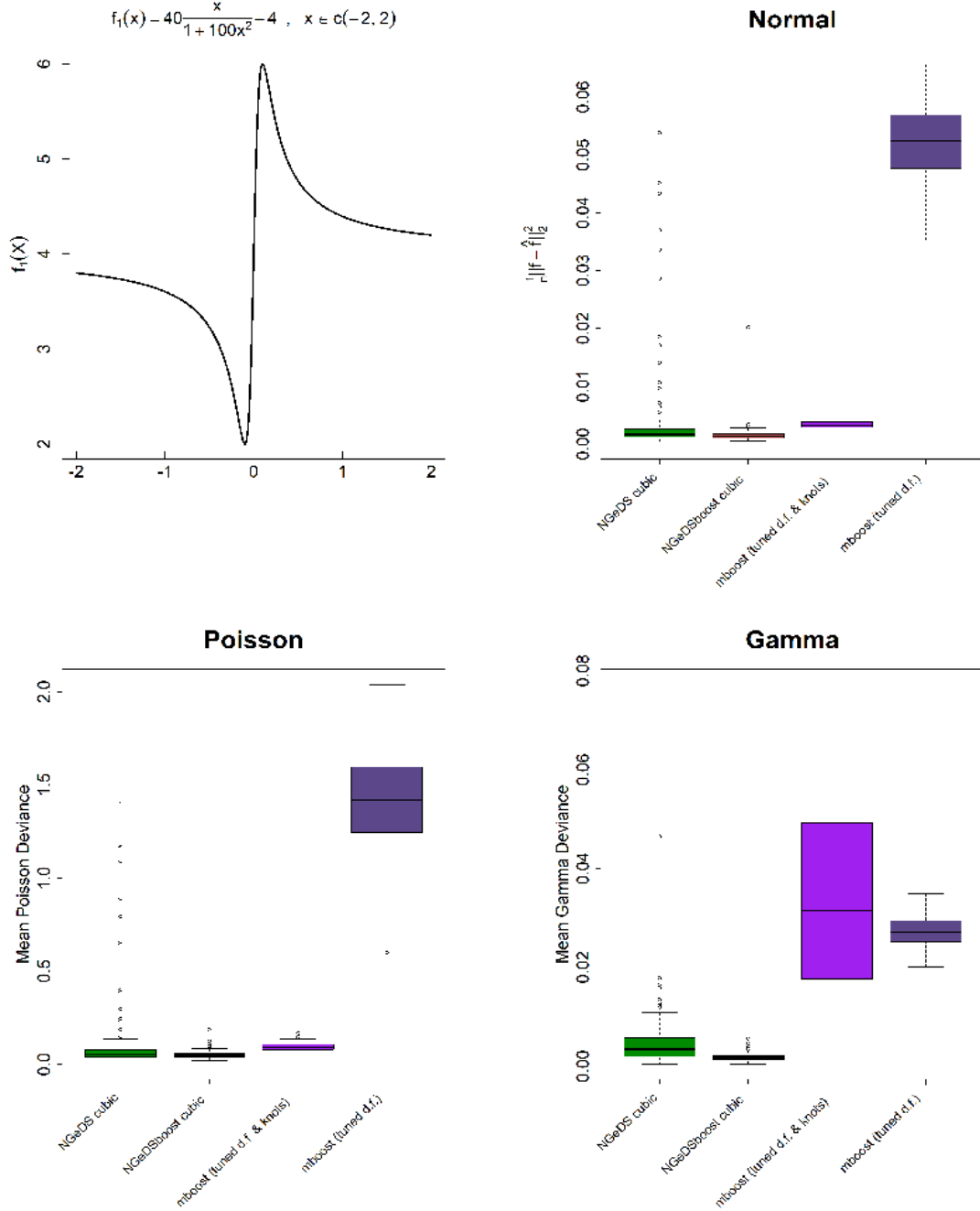


Figure 2: Cubic `NGeDS()`, cubic `NGeDSboost()` and `mboost()` fits over 100 different simulated datasets of example 6.1. The original function, $f_1(x)$, is depicted in black on the top left corner.

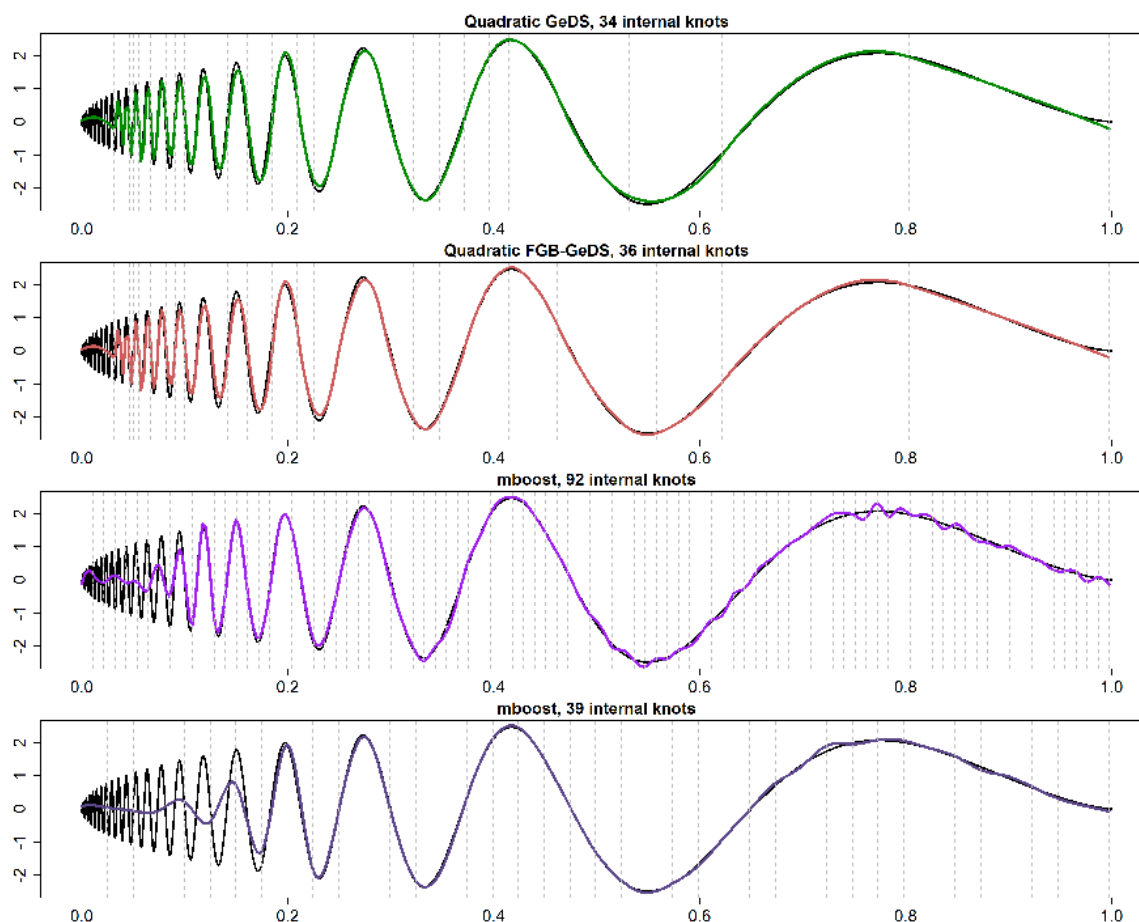


Figure 3: Quadratic `NGeDS()` and quadratic `NGeDSboost()` fits, and `mboost()` fits over a simulated dataset of example 6.2 - Doppler function. The original function, $f_2(x)$, is depicted in black. The dotted vertical lines denote the locations of the knots.

Next, a real data application is considered.

Example 6.4 - BaFe₂As₂. Real data example with $N = 1,151$ from a superconductivity study of Barium-Ferrum-Arsenide (BaFe₂As₂) 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 BaFe₂As₂ 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 6.4 - BaFe2As2
R> data("BaFe2As2")
R> data = BaFe2As2
R> Y <- data$intensity
R> X <- data$angle
R> data <- data.frame(X,Y)
```

For Example 6.4, the GeDS-based fits are run with the example-specific settings `beta = 0.6`, `phi = 0.995` and `q = 3`; all remaining `NGeDS()` and `NGeDSboost()` arguments are kept at their package defaults. Given that the primary objective here is to efficiently capture the signal in the data without overfitting, we fix the number of internal knots in `mboost()` to match that selected by `NGeDSboost()` and tune only the d.f. Since the optimal `NGeDS()` and `NGeDSboost()` fits in this case are 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 2, 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 capture the intensity peaks more closely in this example.

	Type	MSE	Boosting iter.	Internal knots	Time (sec.)
Example 6.4					
<code>NGeDS()</code>	Linear	59,385.34		282	3.78
	Quadratic	70,817.56			
	Cubic	103,247.37			
<code>NGeDSboost()</code>	Linear	59,370.17	3	284	3.44
	Quadratic	70,655.68			
	Cubic	103,079.00			
<code>mboost()</code>	Tuned d.f. & degree	1,563,342.22	10,000	284	1.20

Table 2: Empirical mean squared error, boosting iterations, number of internal knots and computation time for `NGeDS()`, `NGeDSboost()` and `mboost()` for the fits on the real data example 6.4.

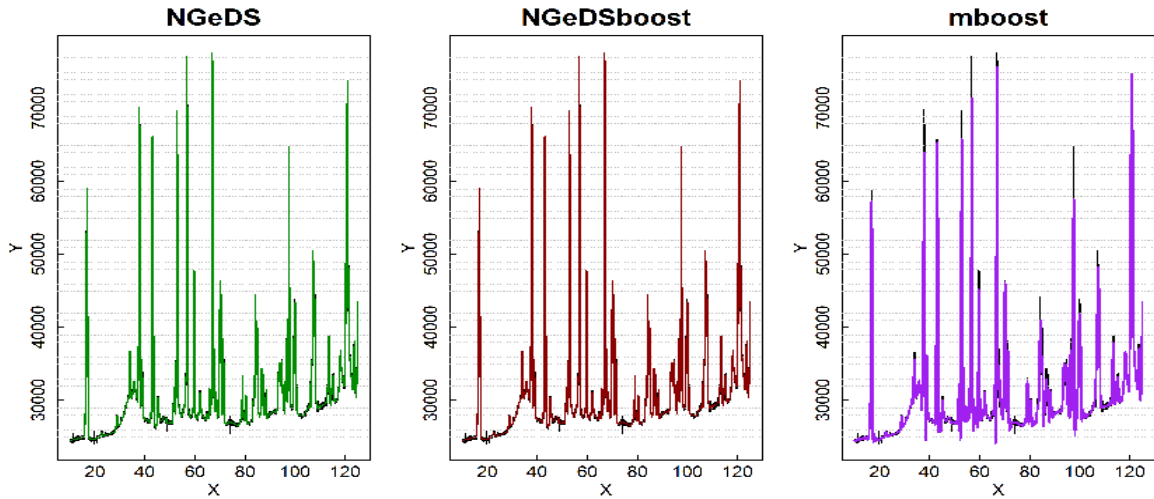


Figure 4: Linear `NGeDS()`, linear `NGeDSboost()` and `mboost()` fits over example 6.4 - BaFe_2As_2 . The original data is depicted in black.

7. Numerical examples – generalized additive models

The two main packages in R for fitting generalized additive models are `gam` (Hastie 2025) and

mgcv (Wood 2025). On the one hand, **gam** follows the theory outlined in the seminal work of 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 extends 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, each with an associated penalty controlling function smoothness. Estimation is then 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).

Both SAS and Stata offer implementations of GAM based on the original procedure of Hastie and Tibshirani. PROC GAM in SAS offers univariate and bivariate thin-plate spline as well as local-regression smoothers, whereas Stata’s **gam** add-on fits univariate cubic smoothing splines. In Python, **pyGAM** (Servén and Brummitt 2018) fits generalized additive models via penalized iteratively reweighted least squares based on penalized splines and supports automatic smoothness selection via grid search, closely mirroring the broad approach of **mgcv**. Since these implementations broadly adhere to the same methodological principles as **gam** and **mgcv**, we do not include them in the comparisons.

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. In Examples 7.1–7.3, the parameters of `NGeDSgam()` are adjusted to avoid overfitting in comparatively dispersed data settings, using lower non-default values of selected stopping-rule parameters. In contrast, Examples 7.4 and 7.5 are run with the default tuning parameters.

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). A complete description of the data is provided in Chambers (1983).

We follow the implementation of this example as presented in Hastie (2025) and correspondingly adapt it for **mgcv** and **GeDS**. This is time-series data. Thus, to compare the performance

of the models, they 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()`, `phi`, which specifies the Stage A stopping-rule threshold for the GeDS smoothers (cf. (3)), is set to 0.7, below its default of 0.99, to stop knot insertion earlier and reduce overfitting. The training and test MSEs are plotted in Figure 5, where it can be seen that the linear `NGeDSgam()` achieves lower test MSEs than the other models across the splitting ratios considered.

Examples 7.2 and 7.3 - Kyphosis. The second and third 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.
- **Start:** The number of the first (topmost) vertebra operated on, also a numeric vector.

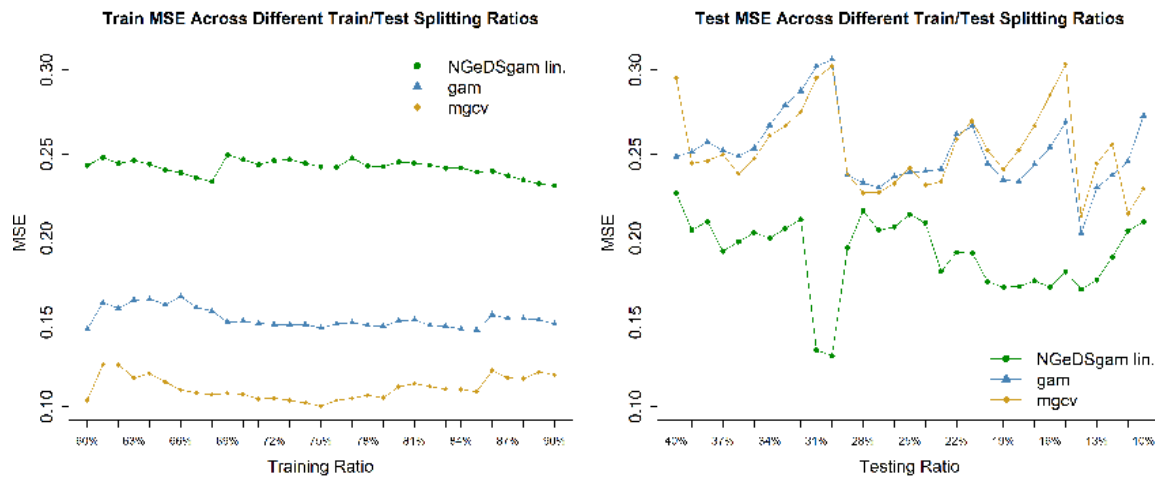


Figure 5: Train (on the left) and test (on the right) MSE across various train/test sample splitting ratios for Example 7.1. Models compared are linear `NGeDSgam()`, `gam` and `mgcv`.

These are binary classification examples, where the objective is to determine the presence or absence of kyphosis using the information provided by the features. In Example 7.2 the covariates considered are `Number` and `Age`, while in Example 7.3, `Age` and `Start` are considered instead. For Example 7.3, the data are additionally restricted to observations with `Number > 2`. The models are compared according to the test binomial deviance, using three train/test splitting ratios: 65%/35%, 70%/30% and 75%/25%. For each splitting ratio, 100 different train/test splits are simulated. 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, q_gam = 1, 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 use non-default tuning parameters in Examples 7.2 and 7.3: the GeDS smoother stopping-rule parameter is set to `phi = 0.7`, and the local-scoring stopping-rule parameter is set to `q_gam = 1`. 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, which provides simulated datasets designed to illustrate the use of `mgcv::gam()`. First, in 7.4, we explore an example from [Gu and Wahba \(1991\)](#) that involves four uniformly distributed covariates. 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 $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$, $\sigma_\epsilon = 0.2$, 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. We refer to the model including the noise predictor x_3 as Example 7.5.1, and to the model excluding x_3 as Example 7.5.2. Consistent with our previous criteria, the MSEs are calculated with respect to the true function. For each example, 100 different datasets were simulated.

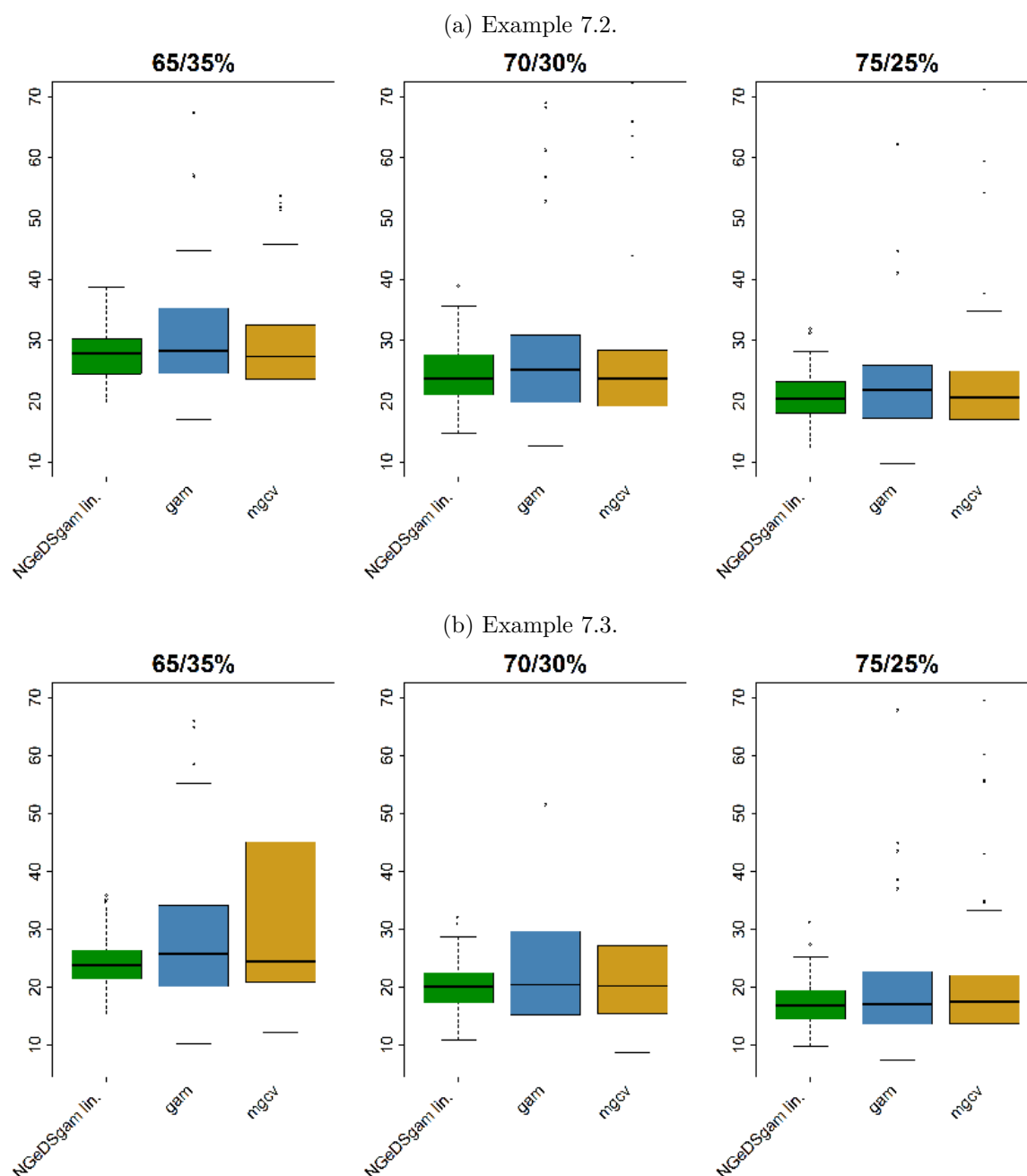


Figure 6: Test binomial deviance for 100 random train/test splits in Examples 7.2 and 7.3, for different train/test splitting ratios (65/35%, 70/30% and 75/25%). Models compared are linear `NGeDSgam()`, `gam` and `mgcv`.

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)
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)
```

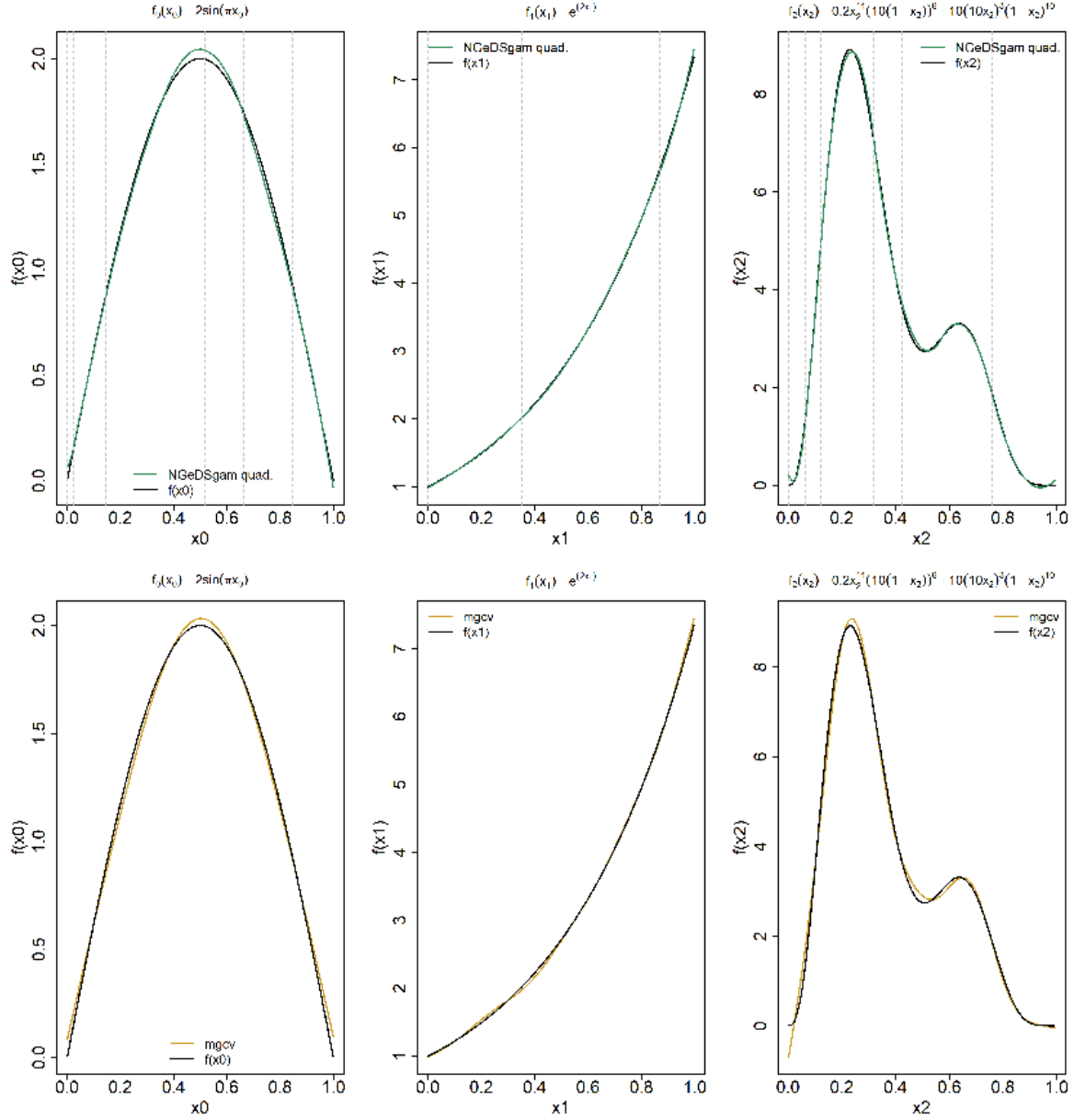


Figure 7: Partial fits from quadratic `NGeDSgam()` and from `mgcv::gam()` on each of the component functions, $f(x_0)$, $f(x_1)$ and $f(x_2)$, that additively form $f(\mathbf{x})$, based on a simulated data sample from Example 7.4. The dotted vertical lines in the GAM-GeDS plots denote the knot locations.

For Examples 7.4, 7.5.1 and 7.5.2, `NGeDSgam()` was run with its default tuning parameters. Figure 8 presents boxplots of the MSEs obtained from 100 simulations for each example.

To keep the scale readable, Figure 8 focuses on `NGeDSgam()` and `mgcv::gam()`. The corresponding `gam::gam()` fits produced substantially larger MSEs in these examples; these values

Model	Example 7.4	Example 7.5.1	Example 7.5.2
NGeDSgam quadratic	0.0043 [0.0035, 0.0053]	0.0073 [0.0062, 0.0086]	0.0067 [0.0053, 0.0078]
NGeDSgam cubic	0.0039 [0.0031, 0.0051]	0.0070 [0.0058, 0.0084]	0.0064 [0.0048, 0.0076]
gam	0.7954 [0.7846, 0.8063]	0.7769 [0.7493, 0.7992]	0.7836 [0.7587, 0.8061]
mgcv	0.0220 [0.0207, 0.0240]	0.0221 [0.0198, 0.0245]	0.0220 [0.0198, 0.0244]

Table 3: Median mean squared error (MSE), with first and third quartiles reported in square brackets, for the GAM comparisons in Examples 7.4, 7.5.1 and 7.5.2.

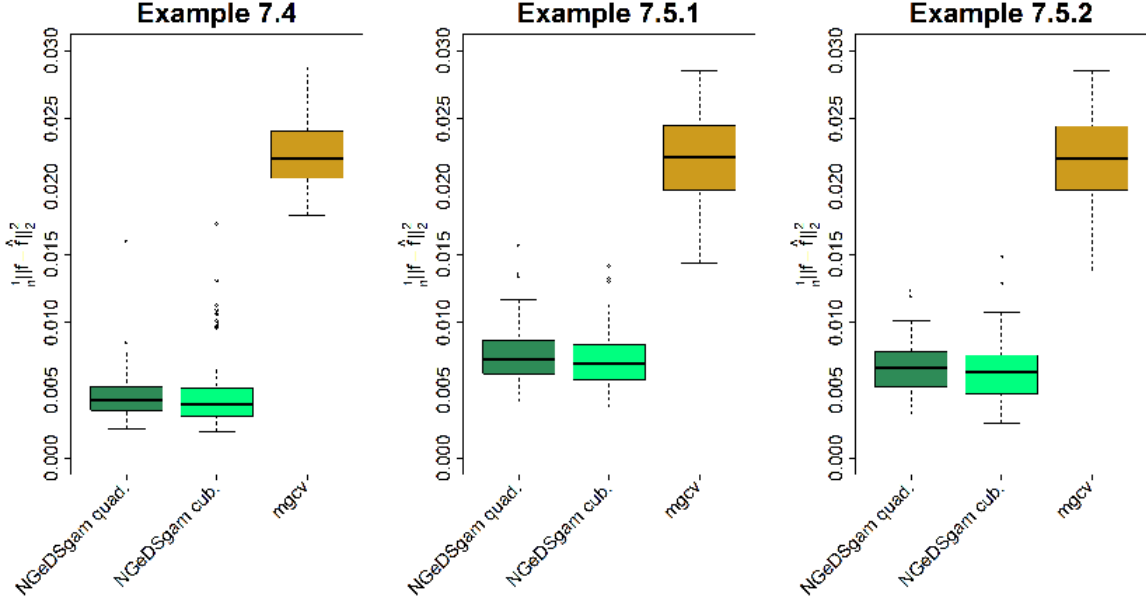


Figure 8: Boxplots of the MSE for quadratic and cubic `NGeDSgam()` and for `mgcv::gam()` fits based on 100 simulations from Examples 7.4, 7.5.1 and 7.5.2. MSE is calculated with respect to the simulated data without noise, i.e., the true generating function.

are reported in Table 3 and are omitted from the boxplots. The boxplots and the corresponding summary table indicate that the quadratic and cubic `NGeDSgam()` fits yield lower MSEs than the `mgcv` and `gam` fits in these examples. Figure 7 illustrates the partial fits for one simulated dataset using quadratic `NGeDSgam()` and using `mgcv::gam()`. While both models recover $f(x_0)$ and $f(x_1)$ well in this simulated sample, the quadratic `NGeDSgam()` fit appears to follow the more intricate structure of $f(x_2)$ more closely. This may help explain the lower MSEs observed for GAM-GeDS in Figure 8 and Table 3.

8. Numerical examples – component-wise boosted models

Bühlmann and Yu (2003) demonstrate that their L_2 Boosting approach can be competitive with standard estimation techniques for fitting additive models, such as backfitting, and may outperform them in some settings. We now present two examples included in the `mboost` documentation (Hothorn *et al.* 2024) and compare the behavior of `NGeDSboost()` with the corresponding `mboost` fits under the specifications considered. 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 the `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 = 100, 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)
```

`NGeDSboost()` is run with a non-default parametrization. Setting `initial_learner = FALSE` uses the empirical risk minimizer as initial learner instead of an initial GeD spline fit. The strength of the GeDS base-learners is reduced by setting `phi = 0.9`, `q = 1` and `shrinkage = 0.6`. Since only the linear `NGeDSboost()` fit is used in this comparison, `higher_order = FALSE` is also specified, so that quadratic and cubic fits are not unnecessarily computed.

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 random splits are generated. Under the suggested parametrization, linear `NGeDSboost()` yields lower median test MSEs than `mboost()` for each of the three train/test splitting ratios considered as can be seen in Figure 9.

Example 8.2 - Synthetic data with 4 predictors. The second example consists of a synthetic dataset of 100 observations with four predictors: two are continuous (x_1 and x_2),

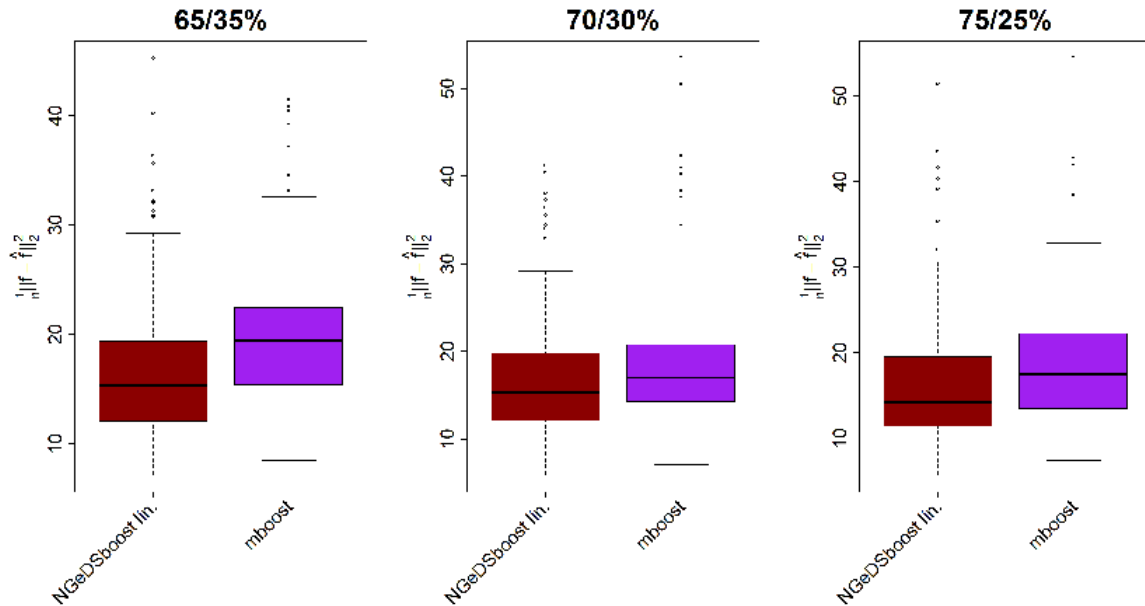


Figure 9: Example 8.1. Test MSE for 100 random train/test splits for different train/test splitting ratios (65/35%, 70/30% and 75/25%). Models compared are linear `NGeDSboost()` and `mboost()`.

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 $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$, $\sigma_\epsilon = 1$. Additionally, observation weights are pre-defined. 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> data <- data.frame(y, x1, x2, x3, x4)
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)
```

For `NGeDSboost()`, we change the GeDS base-learner stopping-rule parameter from its default value to `phi = 0.9`, thereby reducing the strength of the GeDS base-learners, $f(x_1)$ and $f(x_2)$. All remaining tuning parameters are kept at their defaults.

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, with `phi = 0.9`, the quadratic and cubic `NGeDSboost()` fits yield lower MSEs than `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, Jr., 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))
```

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

Therefore a like-for-like spline-boosting comparison between `NGeDSboost()` and `gamboost()` cannot be provided for this example. The subsequent comparison is thus made with the `glmboost()` implementation using linear base-learners, while `NGeDSboost()` is fitted using GeD spline 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)
```

Test binomial deviance boxplots based on 100 random data splits for a train/test split of 70%/30% are included on the right panel of Figure 10. The linear `NGeDSboost()` fit yields lower test binomial deviances than the `glmboost()` implementation considered, bearing in mind that the latter uses linear base-learners whereas `NGeDSboost()` uses spline base-learners. For this high-dimensional binary classification example, `NGeDSboost()` is run with a non-default parametrization. Similar to previous examples, the strength of the GeDS base-learners is reduced by setting `shrinkage = 0.1` and `phi = 0.4`, while a looser boosting stopping rule is employed by fixing `phi_boost_exit = 0.95`. In addition, `initial_learner = FALSE`, meaning that the corresponding empirical risk minimizer is used as initial learner. Since only the linear `NGeDSboost()` fit is used in this comparison, `higher_order = FALSE` is specified.

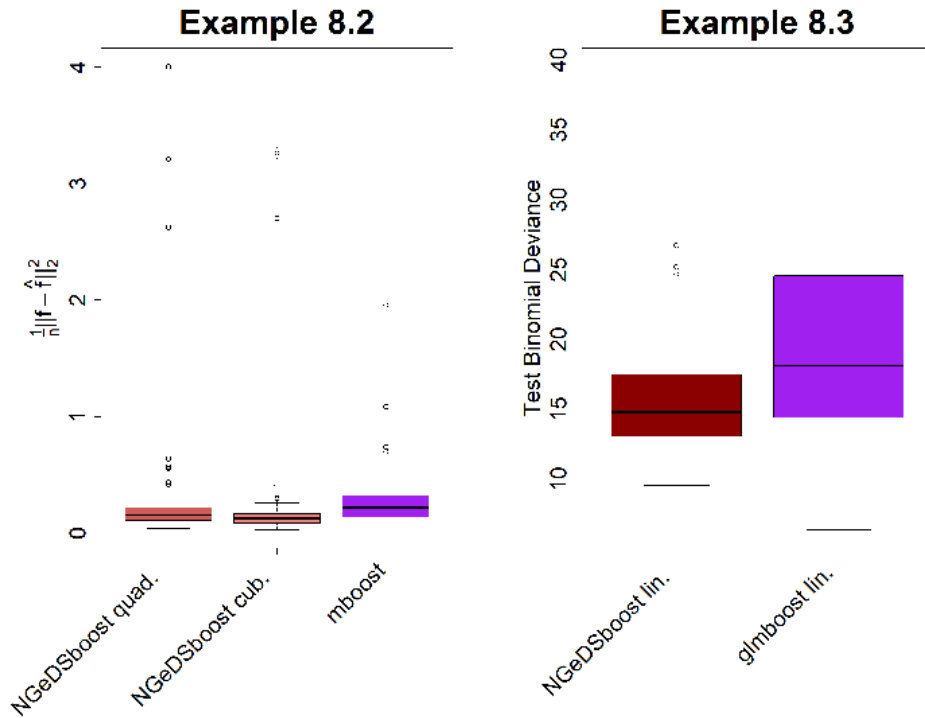


Figure 10: (Left) MSE boxplots for quadratic and cubic `NGeDSboost()` and for `mboost()` fits on 100 simulated data samples of Example 8.2. (Right) Boxplots of the test binomial deviance for linear `NGeDSboost()` and `glmboost()` fits on 100 random data splits of Example 8.3, using a training/test ratio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

9. Conclusions

In this paper, the GeDS methodology, introduced by [Kaishev *et al.* \(2016\)](#) and [Dimitrova *et al.* \(2023\)](#), is significantly extended to include 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. A distinctive feature of the proposed FGB-GeDS approach is that 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 stopping rule based on consecutive deviances, which provides an automatic alternative to cross-validation or AIC-based selection procedures. In the non-additive examples considered, the proposed method yields competitive results relative to the benchmark specifications studied and, in several cases, improves on the 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 implementations offer flexibility in model estimation by 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 tuned via two primary parameters, `phi` and `beta`, with the addition of `phi_gam_exit` and `q_gam` in GAM-GeDS, and `phi_boost_exit` and `q_boost` in FGB-GeDS. The numerical examples considered illustrate that the three approaches can provide accurate fits for more complex univariate functions, as well as for multivariate additive problems. Overall, the numerical examples suggest that quadratic and cubic GeDS-based fits are well suited to the intricate functions considered in this paper. Meanwhile, linear GeDS fits performed well in the sparser-data and high-dimensional examples considered.

In conclusion, GeDS methodology—originally developed for the Normal univariate case and later extended to the GNM framework—is extended to the context of generalized additive models and functional gradient boosting. The examples indicate competitive performance in terms of accuracy and efficiency, together with useful structural properties such as the representation of the final boosted fit as a single spline model.

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., panel data.

Computational details

The results in this paper, displayed in Sections 6, 7 and 8, were obtained using R 4.6.0 on a standard PC (Intel(R) Core(TM) Ultra 9 185H @ 2.50 GHz, 64 GB RAM). Main packages used are **GeDS** 0.3.4, **mboost** 2.9-11, **gam** 1.22-7 and **mgcv** 1.9-4. R itself and all packages used are available from the Comprehensive R Archive Network (CRAN) at <https://CRAN.R-project.org/>.

References

- Bischl B, Lang M, Kotthoff L, Schiffner J, Richter J, Studerus E, Casalicchio G, Jones ZM (2016). “**mlr**: Machine Learning in R.” *Journal of Machine Learning Research*, **17**(170), 1–5. URL <https://jmlr.org/papers/v17/15-066.html>.
- 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.
- 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.
- 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>.
- Brockhaus S, Rügamer D, Greven S (2020). “Boosting Functional Regression Models with **FDboost**.” *Journal of Statistical Software*, **94**(10), 1–50. doi:10.18637/jss.v094.i10. URL <https://www.jstatsoft.org/index.php/jss/article/view/v094i10>.
- Bühlmann P (2006). “Boosting for High-Dimensional Linear Models.” *The Annals of Statistics*, **34**(2), 559–583. doi: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.
- Bühlmann P, van de Geer S (2011). *Statistics for High-Dimensional Data: Methods, Theory and Applications*. Springer Series in Statistics, 1 edition. Springer Berlin, Heidelberg. ISBN 978-3-642-20191-2. doi:10.1007/978-3-642-20192-9.
- Bühlmann P, Yu B (2003). “Boosting with the L_2 Loss.” *Journal of the American Statistical Association*, **98**(462), 324–339. doi:10.1198/016214503000125.
- 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.
- Chambers JM (1983). *Graphical Methods for Data Analysis*. 1st edition. Chapman & Hall/CRC. doi:10.1201/9781351072304.

- Chen T, Guestrin C (2016). “**XGBoost**: A Scalable Tree Boosting System.” In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’16, pp. 785–794. ACM. doi:10.1145/2939672.2939785.
- Dimitrova DS, Kaishev VK, Lattuada A, Sáenz Guillén EL, Verrall RJ (2026). **GeDS**: Geometrically Designed Spline Regression. doi:10.32614/CRAN.package.GeDS. R package version 0.3.4, URL <https://CRAN.R-project.org/package=GeDS>.
- Dimitrova DS, Kaishev VK, Lattuada A, Verrall RJ (2023). “Geometrically designed variable knot splines in generalized (non-)linear models.” *Applied Mathematics and Computation*, **436**, 127493. 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>.
- Dorogush AV, Ershov V, Gulin A (2018). “**CatBoost**: Gradient Boosting with Categorical Features Support.” doi:10.48550/arXiv.1810.11363. URL <https://arxiv.org/abs/1810.11363>.
- Eilers PHC, Marx BD (1996). “Flexible Smoothing with B-Splines and Penalties.” *Statistical Science*, **11**(2), 89–121. doi:10.1214/ss/1038425655.
- Freund Y, Schapire RE (1996). “Experiments with a New Boosting Algorithm.” In *Proceedings of the Thirteenth International Conference on Machine Learning*, ICML’96, pp. 148–156. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 1558604197.
- 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.
- Friedman JH (2001). “Greedy Function Approximation: A Gradient Boosting Machine.” *The Annals of Statistics*, **29**(5), 1189–1232. doi:10.1214/aos/1013203451.
- Garcia AL, Wagner K, Hothorn T, Koenig C, Zunft HJF, Tripp 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 <https://pubmed.ncbi.nlm.nih.gov/15833949/>.
- Gu C (2014). “Smoothing Spline ANOVA Models: R Package **gss**.” *Journal of Statistical Software*, **58**(5), 1–25. doi:10.18637/jss.v058.i05. URL <https://www.jstatsoft.org/index.php/jss/article/view/v058i05>.
- Gu C, Wahba G (1991). “Minimizing GCV/GML Scores with Multiple Smoothing Parameters via the Newton Method.” *SIAM Journal on Scientific and Statistical Computing*, **12**(2), 383–398. doi:10.1137/0912021.

- Hastie T (2007). “Comment: Boosting Algorithms: Regularization, Prediction and Model Fitting.” *Statistical Science*, **22**(4), 513–515. ISSN 08834237. doi:10.1214/07-STS242A. URL <https://www.jstor.org/stable/27645856>.
- Hastie T (2025). *gam: Generalized Additive Models*. doi:10.32614/CRAN.package.gam. R package version 1.22-7, URL <https://CRAN.R-project.org/package=gam>.
- Hastie T, Tibshirani R (1986). “Generalized Additive Models.” *Statistical Science*, **1**(3), 297–318. ISSN 08834237. doi:10.1214/ss/1177013604. URL <https://www.jstor.org/stable/2245459>.
- Hastie T, Tibshirani R (1993). “Varying-Coefficient Models.” *Journal of the Royal Statistical Society: Series B (Methodological)*, **55**(4), 757–779. doi:10.1111/j.2517-6161.1993.tb01939.x. <https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/j.2517-6161.1993.tb01939.x>, URL <https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1993.tb01939.x>.
- Hastie T, Tibshirani R, Friedman JH (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics, 2nd edition. Springer, New York, NY. ISBN 978-0-387-84857-0. doi:10.1007/978-0-387-84858-7.
- Hastie TJ, Tibshirani RJ (1990). *Generalized Additive Models*, volume 43 of *Mono-graphs on Statistics and Applied Probability*. Chapman & Hall, London. doi:10.1201/9780203753781.
- 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. doi:10.1007/s00180-012-0382-5.
- 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 (2024). **mboost**: *Model-Based Boosting*. doi:10.32614/CRAN.package.mboost. R package version 2.9-11, 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.
- Kaishev VK, Dimitrova DS, Haberman S, Verrall RJ (2006a). “Geometrically Designed, Variable Knot Regression Splines: Asymptotics and Inference.” *Working Paper Statistical Research Paper No. 28*, City University London, London, UK. URL <https://openaccess.city.ac.uk/id/eprint/2372/>.
- Kaishev VK, Dimitrova DS, Haberman S, Verrall RJ (2006b). “Geometrically Designed, Variable Knot Regression Splines: Variation Diminish Optimality of Knots.” *Working Paper Statistical Research Paper No. 29*, City University London, London, UK. URL <https://openaccess.city.ac.uk/id/eprint/2373/>.

- 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.
- 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: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. doi:10.1111/1467-9469.00333. <https://onlinelibrary.wiley.com/doi/pdf/10.1111/1467-9469.00333>, URL <https://onlinelibrary.wiley.com/doi/abs/10.1111/1467-9469.00333>.
- Ke G, Meng Q, Finley T, Wang T, Chen W, Ma W, Ye Q, Liu TY (2017). “**LightGBM**: A Highly Efficient Gradient Boosting Decision Tree.” In I Guyon, U von Luxburg, S Bengio, H Wallach, R Fergus, SVN Vishwanathan, R Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30, pp. 3146–3154. Curran Associates, Inc. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.
- 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. doi:10.1038/nmat2443.
- Knafl GJ, Ding K (2016). “Generalized Additive Modeling in SAS.” In *Adaptive Regression for Modeling Nonlinear Relationships*, pp. 315–327. Springer International Publishing, Cham. ISBN 978-3-319-33946-7. doi:10.1007/978-3-319-33946-7_17.
- Koenker R (2005). *Quantile Regression*, volume 38 of *Econometric Society Monographs*. Cambridge University Press, Cambridge. ISBN 978-0-521-84573-1. doi:10.1017/CB09780511754098.
- Kuhn M (2008). “Building Predictive Models in R Using the **caret** Package.” *Journal of Statistical Software*, **28**(5), 1–26. doi:10.18637/jss.v028.i05. URL <https://www.jstatsoft.org/index.php/jss/article/view/v028i05>.
- Kuhn M, Wickham H (2020). *tidymodels: A Collection of Packages for Modeling and Machine Learning Using tidyverse Principles*. URL <https://www.tidymodels.org>.
- Lang M, Binder M, Richter J, Schratz P, Pfisterer F, Coors S, Au Q, Casalicchio G, Kotthoff L, Bischl B (2019). “**mlr3**: A Modern Object-Oriented Machine Learning Framework in R.” *Journal of Open Source Software*, **4**(44), 1903. doi:10.21105/joss.01903. URL <https://joss.theoj.org/papers/10.21105/joss.01903>.
- 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.

- 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.
- 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.
- Opsomer JD (2000). “Asymptotic Properties of Backfitting Estimators.” *Journal of Multivariate Analysis*, **73**(2), 166–179. ISSN 0047-259X. doi:10.1006/jmva.1999.1868. URL <https://www.sciencedirect.com/science/article/pii/S0047259X99918687>.
- Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, VanderPlas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay É (2011). “**scikit-learn**: Machine Learning in Python.” *Journal of Machine Learning Research*, **12**, 2825–2830. URL <https://jmlr.org/papers/v12/pedregosa11a.html>.
- Potts S, Bergherr E, Reinke C, Griesbach C (2024). “Prediction-Based Variable Selection for Component-Wise Gradient Boosting.” *The International Journal of Biostatistics*, **20**(1), 293–314. ISSN 1557-4679. doi:10.1515/ijb-2023-0052.
- Ridgeway G, GBM Developers (2024). **gbm**: *Generalized Boosted Regression Models*. doi:10.32614/CRAN.package.gbm. R package version 2.2.2, URL <https://CRAN.R-project.org/package=gbm>.
- Royston P, Ambler G (2002). “GAM: Stata Module for Generalised Additive Models.” Statistical Software Components, Boston College Department of Economics. URL <https://ideas.repec.org/c/boc/bocode/s428701.html>.
- SAS Institute Inc (2018). *SAS/STAT® 15.1 User’s Guide*. SAS Institute Inc., Cary, NC. The GAM Procedure, URL <https://support.sas.com/documentation/onlinedoc/stat/151/gam.pdf>.
- Schapire RE (1990). “The Strength of Weak Learnability.” *Machine Learning*, **5**(2), 197–227. doi:10.1007/BF00116037.
- 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: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.
- Seabold S, Perktold J (2010). “**statsmodels**: Econometric and Statistical Modeling with Python.” In *Proceedings of the 9th Python in Science Conference*, pp. 92–96. doi:10.25080/Majora-92bf1922-011.
- Servén D, Brummitt C (2018). “**pyGAM**: Generalized Additive Models in Python.” doi:10.5281/zenodo.1208724. URL <https://zenodo.org/records/1208724>.

- Smith NJ, Wardrop M, Capretto T (2024). **patsy**: *Describing Statistical Models in Python Using Symbolic Formulas*. doi:10.5281/zenodo.592075. Python package version 1.0.1, URL <https://pypi.org/project/patsy/>.
- Stasinopoulos DM, Rigby RA (2007). “Generalized Additive Models for Location Scale and Shape (GAMLSS) in R.” *Journal of Statistical Software*, **23**(7), 1–46. doi:10.18637/jss.v023.i07. URL <https://www.jstatsoft.org/index.php/jss/article/view/v023i07>.
- StataCorp (2025). *Stata 19 Base Reference Manual*. Stata Press, College Station, TX. URL <https://www.stata.com/manuals/r.pdf>.
- Wahba G, Wang Y, Gu C, Klein R, Klein B (1995). “Smoothing Spline ANOVA for Exponential Families, with Application to the Wisconsin Epidemiological Study of Diabetic Retinopathy: The 1994 Neyman Memorial Lecture.” *The Annals of Statistics*, **23**(6), 1865–1895. doi:10.1214/aos/1034713638.
- Wand M (2018). **SemiPar**: *Semiparametric Regression*. doi:10.32614/CRAN.package.SemiPar. R package version 1.0-4.2, URL <https://CRAN.R-project.org/package=SemiPar>.
- Wardrop M (2024). **formulaic**: *A High-Performance Implementation of Wilkinson Formulas for Python*. Python package version 1.1.1, URL <https://pypi.org/project/formulaic/>.
- West M, Blanchette C, Dressman H, Huang E, Ishida S, Spang R, Zuzan H, Jr JAO, Marks JR, Nevins JR (2001). “Predicting the Clinical Status of Human Breast Cancer by Using Gene Expression Profiles.” *Proceedings of the National Academy of Sciences of the United States of America*, **98**(20), 11462–11467. doi:10.1073/pnas.201162998.
- Wood SN (2017). *Generalized Additive Models: An Introduction with R*. 2nd edition. Chapman & Hall/CRC, New York. doi:10.1201/9781315370279.
- Wood SN (2025). **mgcv**: *Mixed GAM Computation Vehicle with Automatic Smoothness Estimation*. doi:10.32614/CRAN.package.mgcv. R package version 1.9-4, 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: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.
- Yee TW (2010). “The **VGAM** Package for Categorical Data Analysis.” *Journal of Statistical Software*, **32**(10), 1–34. doi:10.18637/jss.v032.i10. URL <https://www.jstatsoft.org/index.php/jss/article/view/v032i10>.
- Yee TW (2015). *Vector Generalized Linear and Additive Models: With an Implementation in R*. Springer Series in Statistics. Springer New York, New York, NY. ISBN 978-1-4939-2817-0. doi:10.1007/978-1-4939-2818-7.

- 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: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. doi:10.1198/016214501750332820.

A. GeDS Boost model update

In Algorithm 3 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 5 Linear GeDS Boost fit polynomial coefficients update

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

B. Additional examples

Example 6.3 - Blocks function. We assume that the “true” linear predictor, $\eta = f_3(x)$, where

$$f_3(x) = \sum_{j=1}^{11} h_j \frac{1 + \text{sign}(x - s_j)}{2}, \quad x \in [0, 1], \quad (18)$$

with

$$(h_1, \dots, h_{11}) = (4, -5, 3, -4, 5, -4.2, 2.1, 4.3, -3.1, 2.1, -4.2)$$

and

$$(s_1, \dots, s_{11}) = (0.1, 0.13, 0.15, 0.23, 0.25, 0.40, 0.44, 0.65, 0.76, 0.78, 0.81).$$

This is the well-known Blocks function, commonly used as a benchmark for spatially adaptive smoothing methods (see online supplement to [Kaishev et al. \(2016\)](#)). We simulate random samples, $\{X_i, Y_i\}_{i=1}^N$, with $Y_i \sim \mathcal{N}(\mu_i = f_3(X_i), \sigma^2)$, $\sigma = 0.2$, and $X_i = (i - 1)/(N - 1)$, $i = 1, \dots, N$, $N = 2048$.

```
R> # Example 6.3 - Blocks function
R> # Generate a data sample for the response variable
R> # Y and the single covariate X
R> set.seed(123)
```

```

R> N <- 2048
R> X <- (1:N - 1) / (N - 1)
R> # Define the parameters of the Blocks function
R> h_j <- c(4, -5, 3, -4, 5, -4.2, 2.1, 4.3, -3.1, 2.1, -4.2)
R> s_j <- c(0.1, 0.13, 0.15, 0.23, 0.25, 0.40, 0.44,
+         0.65, 0.76, 0.78, 0.81)
R> # Specify a model for the mean of Y to be given by
R> # the Blocks function
R> f_3 <- function(x) {
+   sapply(x, function(z) sum(h_j * (1 + sign(z - s_j)) / 2))
+ }
R> means <- f_3(X)
R> # Add (Normal) noise to the mean of Y
R> Y <- rnorm(N, means, sd = 0.2)

```

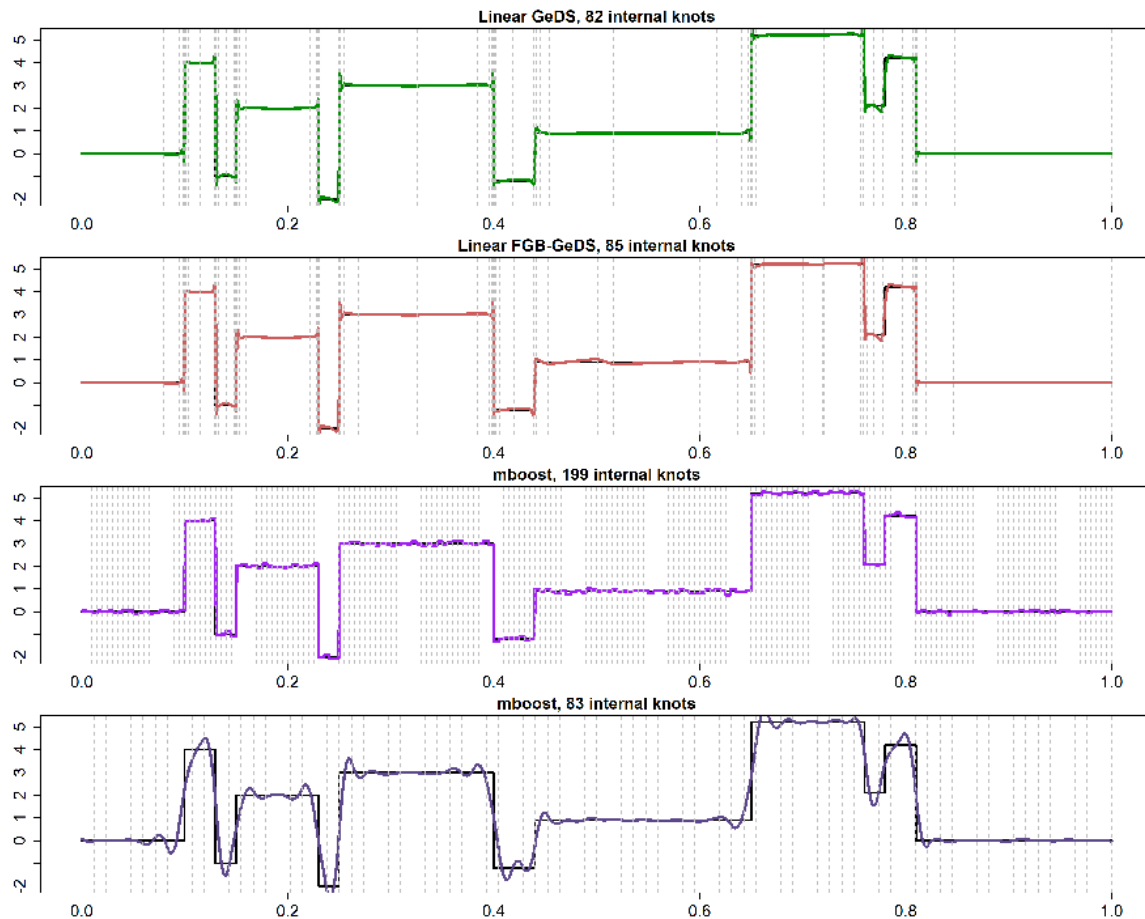


Figure 11: Linear `NGeDS()` and linear `NGeDSboost()` fits, and `mboost()` fits over a simulated dataset of Example 6.3 - Blocks function. The original function, $f_3(x)$, is depicted in black. The dotted vertical lines denote the locations of the knots.

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/dimitrina-dimitrova),

[https://www.bayes.citystgeorges.ac.uk/faculties-and-research/experts/
vladimir-kaishev](https://www.bayes.citystgeorges.ac.uk/faculties-and-research/experts/vladimir-kaishev),

[https://www.bayes.citystgeorges.ac.uk/faculties-and-research/students/
emilio-luis-saenz-guillen](https://www.bayes.citystgeorges.ac.uk/faculties-and-research/students/emilio-luis-saenz-guillen)