Journal of Statistical Software

*August 2020, Volume 94, Issue 8.* *doi: 10.18637/jss.v094.i08*

# BASS: An R Package for Fitting and Performing Sensitivity Analysis of Bayesian Adaptive Spline Surfaces

**Devin Francom**
Los Alamos National Laboratory

**Bruno Sansó**
University of California Santa Cruz

**Abstract**

We present the R package **BASS** as a tool for nonparametric regression. The primary focus of the package is fitting fully Bayesian adaptive spline surface (BASS) models and performing global sensitivity analyses of these models. The BASS framework is similar to that of Bayesian multivariate adaptive regression splines (BMARS) from Denison, Mallick, and Smith (1998), but with many added features. The software is built to efficiently handle significant amounts of data with many continuous or categorical predictors and with functional response. Under our Bayesian framework, most priors are automatic but these can be modified by the user to focus on parsimony and the avoidance of overfitting. If directed to do so, the software uses parallel tempering to improve the reversible jump Markov chain Monte Carlo (RJMCMC) methods used to perform inference. We discuss the implementation of these features and present the performance of **BASS** in a number of analyses of simulated and real data.

*Keywords*: splines, functional data analysis, sensitivity analysis, nonparametric regression.

## 1. Introduction

The purpose of the R (R Core Team 2020) package **BASS** (Francom 2020) is to provide an easy-to-use implementation of Bayesian adaptive spline models for nonparametric regression. It provides a combination of flexibility, scalability, interpretability and probabilistic accuracy that can be difficult to find in other nonparametric regression software packages. The model form is flexible enough to capture local features that may be present in the data. It is scalable to moderately large datasets in both the number of predictors and the number of observations. It performs automatic variable selection. It can build nonparametric functional regression models and incorporate categorical predictors. The package can partition the variability of a resultant model using a nonlinear analysis of variance (ANOVA) decomposition, providing valuable interpretation to the predictors. The Bayesian approach allows for model estimates and predictions that can be evaluated probabilistically. The package is protected under the GNU General Public License, version 3 (GPL-3), and is available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=BASS`.

The BASS framework builds on multivariate adaptive regression splines (MARS) from Friedman (1991b). Well-developed software implementations of the MARS model are available in the R packages **earth** (Milborrow 2019), **polspline** (Kooperberg 2019) and **mda** (Hastie and Tibshirani 2017). The Bayesian version of MARS (BMARS) was first developed in Denison *et al.* (1998). A MATLAB (The MathWorks Inc. 2019) implementation of BMARS is available from the software website accompanying Denison, Holmes, Mallick, and Smith (2002).

Our implementation is more similar to the BMARS implementation, though with some substantial changes to methodology as described in Francom, Sansó, Kupresanin, and Johannesson (2018) and Francom, Sansó, Bulaevskaya, Lucas, and Simpson (2019). The primary motivation for developing this software was building surrogate models (or emulators) for complex and computationally expensive simulators (or computer models). In particular, we wanted to build a fast and accurate surrogate model to use for uncertainty quantification in the presence of a large number of simulations and where each simulation had functional output. Attributing the variance in the response of the surrogate to different combinations of predictors, a practice known as global sensitivity analysis, is a valuable tool for determining which predictors and interactions are important. One of the major benefits of polynomial spline surrogate models is that sensitivity analysis can be done analytically. The **BASS** package has this functionality for scalar and functional response models with both continuous and categorical inputs.

There are a number of other R packages that use splines, such as **crs** (Racine and Nie 2018), **gss** (Gu 2014), **mgcv** (Wood 2017, 2019) and **R2BayesX** (Umlauf, Adler, Kneib, Lang, and Zeileis 2015; Belitz, Brezger, Kneib, Lang, and Umlauf 2017), the latter two of which include possible Bayesian inference methods. These packages allow (or require) the user to specify which variables are allowed to interact in what way, as well as which variables are allowed to have nonlinear main effects. The **crs** package is more similar to the packages that fit MARS models in that it can learn the structure of the model from the data. These packages report a single best model. **BASS** reports an ensemble of models (posterior draws from the model space) that can be used to make probabilistic predictions. In this way, it is more similar to Bayesian nonparametric regression packages like **BART** (McCulloch, Sparapani, Gramacy, Spanbauer, and Pratola 2019) and **tgp** (Gramacy and Taddy 2010).

We introduce the package as follows. In Section 2, we describe the modeling framework, including our methods for posterior sampling, modeling functional responses, and incorporating categorical inputs. In Section 3, we describe the sensitivity analysis methods. Then, in Section 4, we walk through six examples of how to use the package. These are done with **knitr** (Xie 2015) in order to be reproducible by the reader. Finally, in Section 5, we present a summary of the package capabilities.

## 2. Bayesian adaptive spline surfaces

The BASS model, like the MARS and BMARS models, uses data to learn a set of data dependent basis functions that are tensor products of polynomial splines. The number of basis functions as well as the knots and variables used in each basis are chosen adaptively. The BMARS approach uses reversible jump Markov chain Monte Carlo (RJMCMC; Green 1995) to sample the posterior. The BASS adaptation of BMARS includes the improvements of Nott, Kuk, and Duc (2005) for more efficient posterior sampling as well as parallel tempering for better posterior exploration. BASS also efficiently handles functional responses and allows for categorical variables. We discuss each of these aspects below. First, we introduce the BASS model and priors.

Let $y_i$ denote the dependent variable and $\mathbf{x}_i$ denote a vector of $p$ independent variables, with $i = 1, \ldots, n$. Without loss of generality, let each independent variable be scaled to be between zero and one. We model $y_i$ as

$$y_i = f(\mathbf{x}_i) + \epsilon_i, \quad \epsilon_i \sim N(0, \sigma^2) \tag{1}$$

$$f(\mathbf{x}) = a_0 + \sum_{m=1}^{M} a_m B_m(\mathbf{x})$$

$$B_m(\mathbf{x}) = \prod_{k=1}^{K_m} g_{km}[s_{km}(x_{v_{km}} - t_{km})]_+^{\alpha} \tag{2}$$

where $s_{km} \in \{-1, 1\}$ is referred to as a sign, $t_{km} \in [0, 1]$ is a knot, $v_{km}$ selects a variable, $K_m$ is the degree of interaction and $g_{km} = [(s_{km} + 1)/2 - s_{km}t_{km}]^{\alpha}$ is a constant that makes the basis function have a maximum of one. The function $[\cdot]_+$ is defined as $\max(0, \cdot)$. The power $\alpha$ determines the degree of the polynomial splines. We allow for no repeats in $v_{1m}, \ldots, v_{K_m m}$, meaning that a variable can be used only once in each basis function. $M$ is the number of basis functions, and $\mathbf{a}$ is the $M + 1$ vector of basis coefficients (including the intercept). The only difference between this setup and that of MARS and BMARS is the inclusion of the constant $g_{km}$ in each element of the tensor product. This normalizes the basis functions so that the basis coefficients $a_1, \ldots, a_M$ are on the same scale, making computations more stable.

In the course of fitting this model, we seek to estimate $\boldsymbol{\theta} = \{\sigma^2, M, \mathbf{a}, \mathbf{K}, \mathbf{s}, \mathbf{t}, \mathbf{v}\}$ where $\mathbf{K}$ is the $M$-vector of interaction degrees, $\mathbf{s}$ is the vector of signs $\{\{s_{km}\}_{k=1}^{K_m}\}_{m=1}^{M}$, $\mathbf{t}$ the vector of knots and $\mathbf{v}$ the vector of variables used (with $\mathbf{t}$ and $\mathbf{v}$ defined similar to $\mathbf{s}$). Under the Bayesian formulation, we specify a prior distribution for $\boldsymbol{\theta}$.

First, consider the priors for $\sigma^2$ and $\mathbf{a}$. Let $\mathbf{B}$ be the $n \times (M + 1)$ matrix of basis functions (including the intercept). Then we use Zellner's $g$-prior (Liang, Paulo, Molina, Clyde, and Berger 2008) for $\mathbf{a}$ with

$$\mathbf{a}|\sigma^2, \tau, \mathbf{B} \sim N(\mathbf{0}, \sigma^2(\mathbf{B}^\top\mathbf{B})^{-1}/\tau)$$

$$\sigma^2 \sim \mathit{InvGamma}(g_1, g_2)$$

$$\tau \sim \mathit{Gamma}(a_\tau, b_\tau)$$

with default settings $a_\tau = 1/2$ and $b_\tau = 2/n$ (shape and rate), which is the Zellner–Siow Cauchy prior, and $g_1 = g_2 = 0$ resulting in the non-informative prior $p(\sigma^2) \propto 1/\sigma^2$. In practice, the default settings are sufficient for most cases, though it can be helpful to encode actual prior information into the prior for $\sigma^2$.

Now, consider the prior for the number of basis functions, $M$. We use a Poisson prior for $M$, truncated to be between 0 and $M_{\max}$. We give a Gamma hyperprior to the mean of the Poisson, $\lambda$. If $c$ is the Poisson mass that has been truncated, i.e., $c = 1 - \sum_{m=0}^{M_{\max}} e^{-\lambda}\lambda^m/m!$,

| Symbol | $K_{\max}$ | $b$ | $h_1$ | $h_2$ | $g_1$ | $g_2$ | $\alpha$ | $M_{\max}$ | $a_\tau$ | $b_\tau$ |
|---|---|---|---|---|---|---|---|---|---|---|
| `bass` input | `maxInt` | `npart` | `h1` | `h2` | `g1` | `g2` | `degree` | `maxBasis` | `a.tau` | `b.tau` |

Table 1: Translation from mathematical symbols to parameters used in function `bass`.

then we have

$$p(M|\lambda) = \frac{e^{-\lambda}\lambda^M}{cM!} 1(M \leq M_{\max})$$
$$\lambda \sim Gamma(h_1, h_2)$$

where $1(\cdot)$ is the indicator function and the default settings of $h_1 = h_2 = 10$ (shape and rate) in most cases induce a small number of basis functions. In practice, these hyperparameters can be key in order to prevent overfitting. More specifically, we increase $h_2$ (by orders of magnitude in some cases) to bring the prior for $\lambda$ close to zero in an effort to thin out the tails of the Poisson and have fewer basis functions. We use $M_{\max}$ to give an upper bound to the computational cost, rather than to prevent overfitting. This strategy results in better fitting models since setting $M_{\max}$ too small can result in poor RJMCMC mixing. Poor mixing of this sort is due to the fact that the primary way to search the model space with our RJMCMC algorithm is through adding a new basis function, deleting a current basis function, or modifying a current basis function. If we are at the maximum number of basis functions, we can only explore further by modifying the current set of basis functions or deleting basis functions. However, deleting basis functions can be difficult, because they may all be useful enough that deleting one causes a significant drop in the model likelihood. On the other hand, if we are allowed to add a few new basis functions, we may be able to traverse the model space to a different posterior mode, at which point we may be able to delete some of the old basis functions.

The priors for $\mathbf{K}$, $\mathbf{s}$, $\mathbf{t}$ and $\mathbf{v}$ are uniform over a constrained space as described in Francom *et al.* (2018). The constraint in this prior makes sure basis functions have more than $b$ non-zero values. Note that a basis function, as shown in Equation 2, is likely to have many zeros in it depending on how close the knot is to the edge of the space. If a knot is too close to the edge of the space, there might only be a few non-zero values in the basis function. A basis function with only a few non-zero values corresponds to very local fitting and usually results in edge effects (i.e., extreme variance at the edges of the space). If we calculate the number of non-zero points in basis function $m$ to be $b_m$, this prior requires that $b_m > b$. This is the BASS equivalent of specifying a minimum number of points in each partition in recursive partitioning. In addition to specifying $b$, we also specify $K_{\max}$, the maximum degree of interaction for each basis function.

Table 1 shows the parameters used in the function `bass` that we have discussed thus far, and their corresponding mathematical symbols.

## 2.1. Efficient posterior sampling

Posterior sampling is complicated by the fact that the model is transdimensional (since $M$ is unknown). Our RJMCMC scheme allows us to add, delete, or change a basis function consistent with the approach of Nott *et al.* (2005). That is, instead of proposing to add a completely random new basis function in a reversible jump step, we use a proposal generating

| Symbol | $w_1$ | $w_2$ | $N_{\text{MCMC}}$ | $N_{\text{burn}}$ | $N_{\text{thin}}$ |
|---|---|---|---|---|---|
| `bass` input | `w1` | `w2` | `nmcmc` | `nburn` | `thin` |

Table 2: Translation from mathematical symbols to parameters used to specify nominal weights of proposal distributions and number of RJMCMC iterations in function `bass`.

distribution that favors the variables and degrees of interaction already included in the model. For example, say there were four basis functions already in the model, each with degree of interaction two. Say the maximum degree of interaction was three. Then if we were proposing a new basis function we would sample the degree of interaction from $\{1, 2, 3\}$ with weights proportional to $\{w_1, w_1 + 4, w_1\}$, thus favoring two-way interactions since we have seen more of them. If the nominal weight $w_1$ is large compared to the number of basis functions, this distribution looks more uniform. The value $w_2$ is the equivalent nominal weight for sampling variables to be included in a candidate basis function. Both $w_1$ and $w_2$ default to five. If there are a large number of unimportant variables in the data, a small value of $w_2$ (relative to $M$) helps to make posterior sampling more efficient by not proposing basis functions that include the unimportant variables.

We extend the framework of Nott *et al.* (2005) to allow for more than two-way interactions. This ends up being non-trivial, since the RJMCMC acceptance ratio requires us to calculate the probability of sampling the proposed basis function. The difficulty comes when we try to calculate the probability of sampling the particular variables, as this requires calculating the probability of a weighted sample without replacement (weighted because we do not sample variables uniformly, without replacement because variables cannot be used more than once in the same basis function). This is equivalent to sampling from the multivariate Wallenius' noncentral hypergeometric distribution. In earlier versions of **BASS**, we determined the probability of such a sample using a function from the R package **BiasedUrn** (Fog 2015). Since the CRAN version of **BiasedUrn** allows for only 32 possible variables, we included a slightly altered version of the function in **BASS** to quickly evaluate the approximate density function of the multivariate Wallenius' noncentral hypergeometric distribution. In the current version of **BASS**, we have implemented a simplified multivariate Wallenius' noncentral hypergeometric distribution better fit for this specific purpose.

The computation behind posterior sampling becomes much more efficient when we recognize that each RJMCMC iteration only allows slight changes to our set of basis functions. Thus, quantities like $\mathbf{B}^\top\mathbf{B}$, $\mathbf{Ba}$ and $\mathbf{B}^\top\mathbf{y}$ can easily be updated rather than recalculated, as shown in Francom *et al.* (2018).

We perform $N_{\text{MCMC}}$ RJMCMC iterations and discard the first $N_{\text{burn}}$, after which every $N_{\text{thin}}$ iteration is kept. This results in $(N_{\text{MCMC}} - N_{\text{burn}})/N_{\text{thin}}$ posterior samples. Table 2 shows the parameters used in function `bass` discussed in this section, as well as their mathematical symbols.

## 2.2. Parallel tempering

Posterior sampling with RJMCMC is prone to mixing problems (i.e., problems exploring all of the parameter space). In our case, this is because only slight changes to the basis functions can be made in each iteration. Thus, once we start sampling from one mode of the posterior, it can be hard to move to another mode if it requires changing many of the basis functions (Gramacy, Samworth, and King 2010).

| Symbol | $(t_1, \dots, t_T)$ | $N_{st}$ |
|---|---|---|
| `bass` input | `temp.ladder` | `start.temper` |

Table 3: Translation from mathematical symbols to parameters used for parallel tempering in function `bass`.

We are able to achieve better mixing by using parallel tempering. This requires the specification of a temperature ladder, $1 = t_1 < t_2 < \cdots < t_T < \infty$. For each temperature in the temperature ladder, a RJMCMC chain samples the posterior raised to the inverse temperature (i.e., if $\pi(\boldsymbol{\theta}|\mathbf{y})$ is the posterior of interest, we sample from $\pi(\boldsymbol{\theta}|\mathbf{y})^{1/t_i}$). The chains at neighboring temperatures are allowed to swap states according to a Metropolis-Hastings acceptance ratio (see Francom *et al.* 2018 and references therein). Only samples in the lowest temperature chain ($t_1$) are used for inference. The high temperature chains mix over many posterior modes, allowing diverse models to be propagated to the low temperature chain. We allow the chains to run without swapping for $N_{st}$ iterations at the beginning of the run to allow them to get close to their stationary distributions.

Specifying a temperature ladder can be difficult. Temperatures need to be close enough to each other to allow for frequent swaps (with acceptance rates between 20 and 60%; Altekar, Dwarkadas, Huelsenbeck, and Ronquist 2004), and the highest temperature ($t_T$) needs to be high enough to be able to explore all the modes. Future versions of this package may make some attempt at automatically specifying and altering a temperature ladder. Further, a message passing interface (MPI) approach to handling the multiple chains could result in substantial speedup, and may be implemented in future versions of the package.

Table 3 shows the translation from parameters used for parallel tempering in function `bass` to symbols we have used in this section.

## 2.3. Functional response

The **BASS** package has two implementations of methods to use for functional response modeling.

### *Augmentation approach*

The augmentation approach handles functional responses as though the variable indexing the functional response, like time or location, is one of the independent variables. When the functional response is output onto the same functional variable grid for all samples, this results in more efficient calculations involving basis functions because of the Khatri-Rao product structure (Francom *et al.* 2018). For example, this software is well suited to fit a model where the data are such that a combination of independent variables results in a time series and the grid of times (say, $r_1, \dots, r_q$) is the same for each combination.

If there are multiple functional variables, we must specify a maximum degree of interaction for them, $K^F_{\max}$. For instance, if the functional output was a spatiotemporal field (a function of three variables) and we specify a maximum degree of functional interaction of two, we would not allow for interactions between both spatial dimensions and time. We would specify the grid of spatial locations and time points as a matrix with three columns rather than a vector like we did in the time series example above. We can also specify a value $b_F$, possibly different from $b$, that indicates the number of non-zero values required in the functional part

| Symbol | $(r_1, \ldots, r_q)$ | $K^F_{\max}$ | $b_F$ |
|---|---|---|---|
| `bass` input | `xx.func` | `maxInt.func` | `npart.func` |

Table 4: Translation from mathematical symbols to parameters used in function `bass` when modeling functional data.

of basis functions. When functional responses are included, the values of $b$ and $b_F$ should be relative to the sample size and the size of the functional grid, respectively.

Table 4 shows parameters necessary to model functional responses with the augmentation approach in function `bass`. The response `y` should be specified as a matrix when the response is functional.

*Basis expansion approach*

The second approach that the **BASS** package can use for modeling functional responses requires decomposing the functional response onto a set of basis functions $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_{N_u}]$, so that the matrix of functional responses $\mathbf{Y} = \mathbf{U}\boldsymbol{\Phi}$, where each column of $\mathbf{Y}$ is a (flattened) functional response, and $\mathbf{Y}$ has $n$ columns. Hence $\boldsymbol{\Phi}$ is the $N_u \times n$ matrix of coefficients in the basis decomposition. Often, we will center and scale the functional responses using the pointwise mean ($\boldsymbol{\mu}$) and standard deviation ($\boldsymbol{\kappa}$) of the functional responses before taking the basis decomposition. For dimension reduction purposes, we often use only a subset of $n_u \leq N_u$ basis functions, denoted $\mathbf{U}^*$ with corresponding subset coefficients $\boldsymbol{\Phi}^*$. Our approach to functional response modeling is then to fit independent models for the coefficients in the basis decomposition (Francom *et al.* 2019). If $Y_{ij}$ is the response for the $j$th functional variable setting using inputs $\mathbf{x}_i$, this approach models $Y_{ij}$ with

$$Y_{ij} = \sum_{l=1}^{n_u} U^*_{lj}\phi^*_l(\mathbf{x}_i)$$
$$\phi^*_l(\mathbf{x}_i) \sim \text{univariate BASS model}$$

where $\phi^*_l(\mathbf{x}_i) \equiv \phi^*_{li}$ is modeled with the same specification given in Equation 1, so that $\phi^*_{li} = f_l(\mathbf{x}_i) + \epsilon_{li}, \quad \epsilon_{li} \sim N(0, \sigma^2_l)$. These models are fit independently, and thus can be done in parallel. In other words, this approach makes $n_u$ independent calls to the `bass` function.

This approach can often provide more accurate results than the augmentation approach, and if the number of available threads is greater than or equal to $n_u$, this can be done in the same amount of time it takes to fit a single univariate model.

Table 5 shows parameters necessary to model functional responses with the general basis function approach in function `bassBasis` (these parameters are passed as a list). Note that the shortcut function `bassPCA` calculates a principal component basis given the matrix `y` and fits the corresponding models, and thus requires only a subset of these parameters (not passed as a list).

## 2.4. Categorical inputs

We include categorical variables by allowing for basis functions to include indicators for categorical variables being in certain categories. Our approach is the Bayesian version of Friedman

| Symbol | $\mathbf{X}$ | $n_u$ | $\mathbf{U}^*$ | $\Phi^*$ | $\boldsymbol{\mu}$ | $\boldsymbol{\kappa}$ |
|---|---|---|---|---|---|---|
| `bassBasis` input | `xx` | `n.pc` | `basis` | `newy` | `y.m` | `y.s` |

Table 5: Translation from mathematical symbols to parameters used in function `bassBasis` when modeling functional data. A subset of these parameters are used in function `bassPCA`.

(1991a) and is described in Francom *et al.* (2019). If a set of independent variables is separated into continuous variables $\mathbf{x}$ and categorical variables $\mathbf{c}$, then the $m$th basis function equivalent of Equation 2 can be written as

$$B_m(\mathbf{x},\mathbf{c}) = \prod_{k=1}^{K_m} g_{km}[s_{km}(x_{v_{km}} - t_{km})]_+^{\alpha} \prod_{l=1}^{K_m^c} 1\left(c_{v_{lm}^c} \in C_{lm}\right),$$

where $K_m^c$ is the degree of interaction for the categorical predictors, $1(\cdot)$ is the indicator function, $v_{lm}^c$ indexes the categorical variables and $C_{lm}$ is a subset of the categories for variables $c_{v_{lm}^c}$. We now allow for $K_m$ or $K_m^c$ to be zero, and specify a $K_{\max}^c$ (`maxInt.cat` in function `bass`).

The priors we use for the degree of interaction, variables used and categories used are, in combination with the priors we used above, the same constrained uniform ones. Thus, basis function $(B_m(\mathbf{x}_1,\mathbf{c}_1),\ldots,B_m(\mathbf{x}_n,\mathbf{c}_n))$ is required to have at least $b$ non-zero values.

## 3. Sensitivity analysis

Global sensitivity analysis for nonlinear models using the Sobol' decomposition (Sobol' 2001) is well developed, but often requires large numbers of evaluations of the models for Monte Carlo approximation of integrals (Saltelli *et al.* 2008). The benefit of polynomial spline models is that Monte Carlo approximation is unnecessary because the integrals can be calculated analytically.

The method decomposes a function $f(\mathbf{x})$ into main effects, two-way interactions, and so on, up to $p$-way interactions so that

$$f(\mathbf{x}) = f_0 + \sum_{i=1}^{p} f_i(x_i) + \sum_{i=1}^{p}\sum_{j>i} f_{ij}(x_i,x_j) + \cdots + f_{1\cdots p}(x_1,\ldots,x_p). \tag{3}$$

Each term in the sum above is constructed so that it is centered at zero and orthogonal to all the other terms. This can be done by calculating the centered (conditional) expectations

$$\begin{aligned} f_0 &= \int f(\mathbf{x})p(\mathbf{x})d\mathbf{x} \\ f_i(x_i) &= \int f(\mathbf{x})p(\mathbf{x})d\mathbf{x}_{-i} - f_0 \\ f_{ij}(x_i,x_j) &= \int f(\mathbf{x})p(\mathbf{x})d\mathbf{x}_{-ij} - f_i(x_i) - f_j(x_j) - f_0 \end{aligned}$$

etc., for all the terms in Equation 3. It is often assumed that $\mathbf{x}$ is independent uniform distributed, though the **BASS** package allows for more complex distributions, still with the assumption of independence. Since the terms in Equation 3 are orthogonal and zero-centered,

$$\mathsf{E}(f^2(\mathbf{x})) = f_0^2 + \sum_{i=1}^{p} \mathsf{E}\left(f_i^2(x_i)\right) + \sum_{i=1}^{p}\sum_{j>i} \mathsf{E}\left(f_{ij}^2(x_i, x_j)\right) + \cdots + \mathsf{E}\left(f_{1\cdots p}^2(x_1, \ldots, x_p)\right).$$

Using the fact that $\mathsf{VAR}(f(\mathbf{x})) = \mathsf{E}(f^2(\mathbf{x})) - f_0^2$ and that $\mathsf{E}(f_{i_1\cdots i_s}(x_{i_1\cdots i_s})) = 0$ for all terms except $f_0$,

$$\mathsf{VAR}(f(\mathbf{x})) = \sum_{i=1}^{p} \mathsf{VAR}(f_i(x_i)) + \sum_{i=1}^{p}\sum_{j>i} \mathsf{VAR}(f_{ij}(x_i, x_j)) + \cdots + \mathsf{VAR}(f_{1\cdots p}(x_1, \ldots, x_p)).$$

This is a decomposition of the variance of the model into variance due to each main effect, each two-way interaction (after accounting for the associated main effects), etc. Under independent uniform $p(\mathbf{x})$ (as well as some others), all of these integrals are analytical, with solutions given in Francom *et al.* (2018). Sensitivity indices for main effects and interactions are then defined as proportions of the total variance. Total sensitivity for a particular variable can then be gauged by adding the main effect and all interactions associated with that variable and comparing to the total sensitivity indices for other variables.

We can obtain this variance decomposition for each posterior sample to get posterior distributions of sensitivity indices. This can be time consuming, so the `sobol` function has an argument `mcmc.use` to specify which RJMCMC iterations should be used. Calculations of the integrals above can be vectorized when basis functions are the same and only basis function coefficients change. This is the case for many of the RJMCMC iterations, and the `sobol` function automatically determines this and accounts for it. (As a side note, this is also the case for the `predict` function).

## 3.1. Functional response

There are a few ways to think about sensitivity analysis for models with functional response. One way is to get the sensitivity indices for the functional variables in the same way we get the sensitivity indices for the rest of the variables. This results in a total variance decomposition. Another approach is to obtain functional sensitivity indices, which would tell us how important a variable or interaction is as we change the functional variable. This can be done by following the procedure just mentioned, but simply not integrating over the functional variable. Hence, all of the expectations above would be conditional on the functional variable. These approaches are explored in Francom *et al.* (2018) and Francom *et al.* (2019).

By default, function `sobol` gets sensitivity indices for the functional variables the same way it does for the other variables. Setting `func.var = 1` gets the sensitivity indices as functions of the first (possibly only) functional variable (if there are multiple functional variables, this refers to the first column of the matrix `xx.func` passed to function `bass`).

On the other hand, the `sobolBasis` function (used on an object obtained using `bassBasis` or `bassPCA`) gets functional sensitivity indices by default. Because of the complexity of these integrals across the models for the different basis function coefficients, this function can often be sped up significantly by performing the integrals in parallel (using the `n.cores` parameter).

## 3.2. Categorical inputs

Under our categorical input extension, the necessary expectations to obtain the Sobol' decomposition are still analytical, as described in Francom *et al.* (2019). For the categorical variables, we replace the integrals with sums over categories.

## 4. Examples

We now demonstrate the capabilities of the package on a few examples. For each example, we start by setting the seed (`set.seed(0)`) so that readers can replicate the results. First we load the package

```
R> library("BASS")
```

which we use for all the examples.

### 4.1. Curve fitting

We first demonstrate how the package can be used for curve fitting. We generate $y \sim N(f(x), 1)$ where $x \in [-5, 5]$ and

$$f(x) = \begin{cases} -0.1x^3 + 2\sin(\pi x^2)(x-4)^2 & 0 < x < 4 \\ -0.1x^3 & \text{otherwise} \end{cases}$$

for 1000 samples of $x$. The data are shown in Figure 3.

We generate the data with the following code.

```
R> set.seed(0)
R> f <- function(x) {
+    -0.1 * x^3 +
+      2 * as.numeric((x < 4) * (x > 0)) * sin(pi * x^2) * (x - 4)^2
+  }
R> sigma <- 1
R> n <- 1000
R> x <- runif(n, -5, 5)
R> y <- rnorm(n, f(x), sigma)
```

We then call function `bass` to fit a BASS model using the default settings.

```
R> mod <- bass(x, y)

MCMC Start #-- May 07 16:37:09 --# nbasis: 0
MCMC iteration 1000 #-- May 07 16:37:11 --# nbasis: 21
MCMC iteration 2000 #-- May 07 16:37:13 --# nbasis: 21
MCMC iteration 3000 #-- May 07 16:37:15 --# nbasis: 23
MCMC iteration 4000 #-- May 07 16:37:17 --# nbasis: 24
MCMC iteration 5000 #-- May 07 16:37:18 --# nbasis: 22
MCMC iteration 6000 #-- May 07 16:37:20 --# nbasis: 22
MCMC iteration 7000 #-- May 07 16:37:22 --# nbasis: 25
MCMC iteration 8000 #-- May 07 16:37:23 --# nbasis: 22
MCMC iteration 9000 #-- May 07 16:37:25 --# nbasis: 22
MCMC iteration 10000 #-- May 07 16:37:27 --# nbasis: 23
```

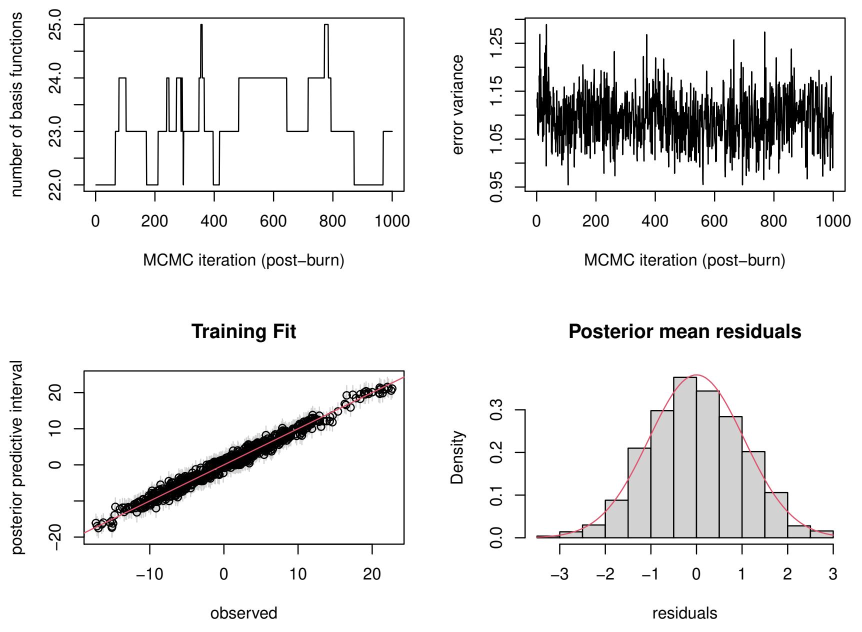

Figure 1: Diagnostic plots for BASS model fitting.

The result is an object that can be used for prediction and sensitivity analysis. By default, `bass` prints progress after each 1000 MCMC iterations, along with the number of basis functions. To diagnose the fit of the model, we call the `plot` function.

```
R> plot(mod)
```

This generates the four plots shown in Figure 1. The top left and right plots show trace plots (after burn-in and excluding thinned samples) of the number of basis functions ($M$) and the error variance ($\sigma^2$). The bottom left plot shows the response values plotted against the posterior mean predictions (with equal tail posterior probability intervals as specified by the `quants` parameter). The bottom right plot shows a histogram of the posterior mean residuals along with the assumed Gaussian distribution centered at zero and with variance taken to be the posterior mean of $\sigma^2$. This is for checking the Normality assumption.

Next, we can generate posterior predictions at new inputs, which we generate as `x.test`.

```
R> n.test <- 1000
R> x.test <- sort(runif(n.test, -5, 5))
R> pred <- predict(mod, x.test, verbose = TRUE)

Predict Start #-- May 07 16:37:28 --# Models: 40
```

By default, the `predict` function generates posterior predictive distributions for all of the inputs. We can use a subset of posterior samples by specifying the parameter `mcmc.use`. For instance, `mcmc.use = 1:5` will use the first five posterior samples (after burn-in and excluding thinned samples), and will thus be faster. Rather than iterating through the MCMC samples to generate predictions, we instead iterate through "models." The model changes when the basis functions change, which means that we can build the basis functions once

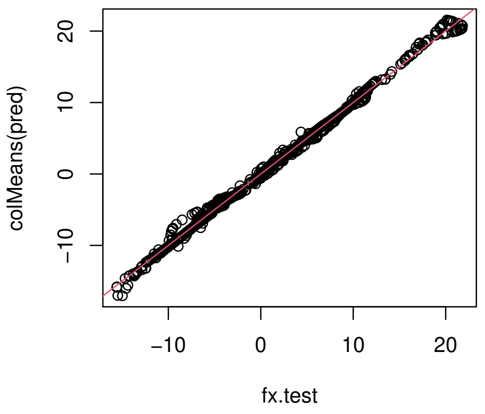

Figure 2: BASS prediction on test data.

and perform vectorized operations for predictions for all the MCMC iterations with the same basis functions.

The object resulting from the `predict` function is a matrix with rows corresponding to MCMC samples and columns corresponding to settings of `x.test`. Thus, the posterior mean predictions are obtained by taking the column means. We plot the posterior predictive means against the true values of $f(x)$ as shown in Figure 2.

```
R> fx.test <- f(x.test)
R> plot(fx.test, colMeans(pred))
R> abline(a = 0, b = 1, col = 2)
```

Note that the predictive distributions in the columns of `pred` are for $f(x)$. To obtain predictive distributions for data, we would need to include Gaussian error with variance $\sigma^2$ (demonstrated in Section 4.5). Posterior samples of $\sigma^2$ are given in `mod$s2`.

In the curve fitting case, we can plot predicted curves. Below, we plot 10 posterior predictive samples along with the true curve (Figure 3). We also show knot locations (in the rug along the $x$-axis) for one of the posterior samples.

```
R> plot(x, y, cex = 0.5)
R> curve(f(x), add = TRUE, lwd = 3, n = 1000, col = 2, lty = 2)
R> matplot(x.test, t(pred[seq(100, 1000, 100), ]), type = "l", add = TRUE,
+    col = 3)
R> rug(BASS:::unscale.range(mod$curr.list[[1]]$knots.des, range(x)))
R> legend("topright", legend = c("true curve", "posterior predictive draws"),
+    col = 2:3, lty = c(2, 1), lwd = c(3, 1), bty = "n")
```

If we are interested in using fewer knots (fewer basis functions), we can change the prior for the number of basis functions to be more restrictive. For instance, setting `h2 = 100`

```
R> mod <- bass(x, y, h2 = 100)
R> pred <- predict(mod, x.test)
R> plot(x, y, cex = 0.5)
```

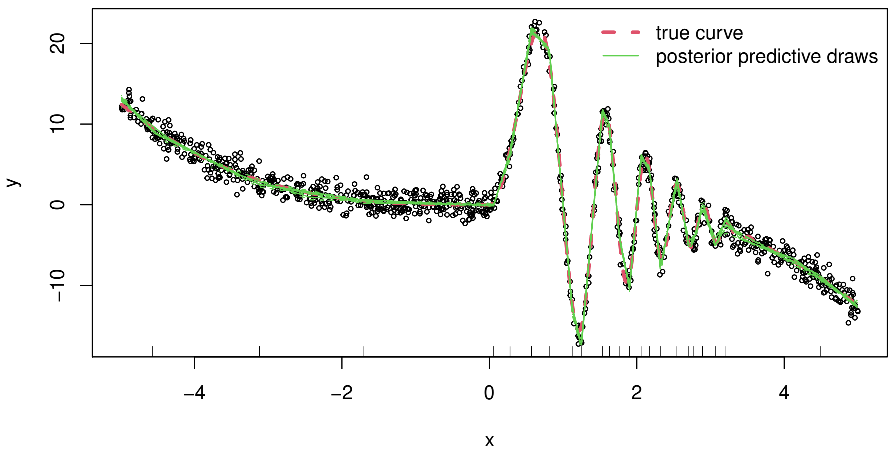

Figure 3: True curve with posterior predictive draws.

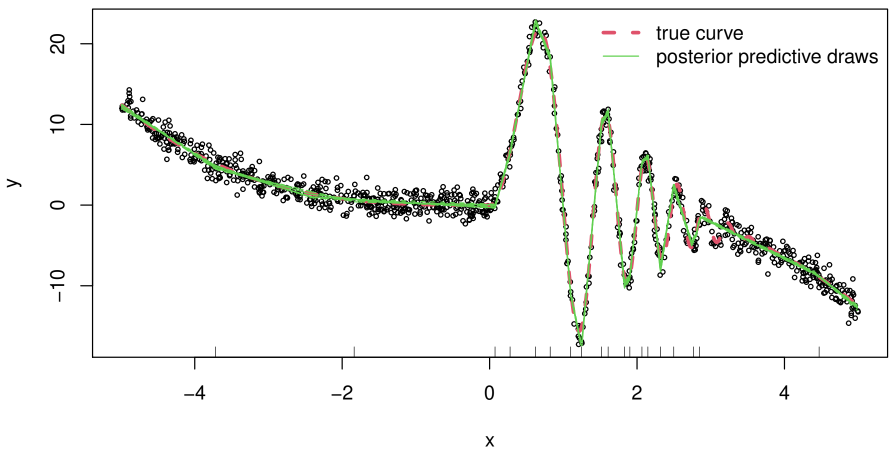

Figure 4: True curve with posterior predictive draws and more restrictive prior on the number of basis functions.

```
R> curve(f(x), add = TRUE, lwd = 3, n = 1000, col = 2, lty = 2)
R> matplot(x.test, t(pred[seq(100, 1000, 100), ]), type = "l", add = TRUE,
+    col = 3)
R> rug(BASS:::unscale.range(mod$curr.list[[1]]$knots.des, range(x)))
R> legend("topright", legend = c("true curve", "posterior predictive draws"),
+    col = 2:3, lty = c(2, 1), lwd = c(3, 1), bty = "n")
```

results in knots as shown along the $x$-axis of Figure 4. This results in fewer knots, but perhaps slight underfitting in the part of the curve around $x = 3$. The `h2` parameter can be used to prevent overfitting, but the setting is not intuitive. Thus, this parameter may require tuning (perhaps by cross-validation).

Two final issues to discuss with this example are why we use linear splines (the default `degree = 1`) and how to tell if we have achieved convergence before taking MCMC samples as posterior samples. We use linear splines almost exclusively when using this package because of their stability and ability to capture nonlinear curves and surfaces. Using a higher degree, such as `degree = 3`, results in smoother models but suffers from stability problems and is more difficult to fit. We suggest settings of `degree` other than `degree = 1` be used with care, always with scrutiny of prediction performance. Convergence is best assessed by examining the trace plots shown in Figure 1. Especially if the trace plot for $\sigma^2$ shows any sort of non-cyclical pattern, the sampler should be run for longer. As a side note, a new sampler can be started from where the old sampler left off by using the `curr.list` parameter. For instance, we can run `mod2 <- bass(x, y, curr.list = mod$curr.list)` to start a new sampler from where `mod` left off.

## 4.2. Friedman function

For our next example, we will test the package on the Friedman function (Friedman 1991b). This function will have 10 inputs, five of which contribute nothing. The other five are used to generate

$$f(\mathbf{x}) = 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5.$$

We generate 200 input samples uniformly from a unit hypercube, calculate $f(x)$ for each and add standard Normal error to obtain data to model.

```
R> set.seed(0)
R> f <- function(x) {
+   10 * sin(pi * x[, 1] * x[, 2]) + 20 * (x[, 3] - 0.5)^2 +
+     10 * x[, 4] + 5 * x[, 5]
+ }
R> sigma <- 1
R> n.vars <- 10
R> n <- 200
R> x <- matrix(runif(n * n.vars), n, n.vars)
R> y <- rnorm(n, f(x), sigma)
```

Here we will show how we can change the length of the MCMC chain and use parallel tempering. We run the RJMCMC chain for 40000 iterations, discarding the first 30000 as burn-in and thinning by keeping every tenth sample. We supply a temperature ladder with smallest value one (the "cold chain", or true posterior) and largest value 11.03 (the "hottest" chain) using geometric spacing. Thus, $t_i = (1 + \Delta_t)^{i-1}$ where $\Delta_t$ is a spacing parameter we set at 0.35. We use nine chains. By default, chains at neighboring temperatures will be allowed to swap after the first 1000 iterations.

```
R> mod <- bass(x, y, nmcmc = 40000, nburn = 30000, thin = 10,
+    temp.ladder = (1 + 0.35)^(1:9 - 1), verbose = FALSE)
```

We can generate posterior predictive samples just as we did in the curve fitting example.

```
R> n.test <- 1000
R> x.test <- matrix(runif(n.test * n.vars), n.test)
R> pred <- predict(mod, x.test, verbose = TRUE)
```

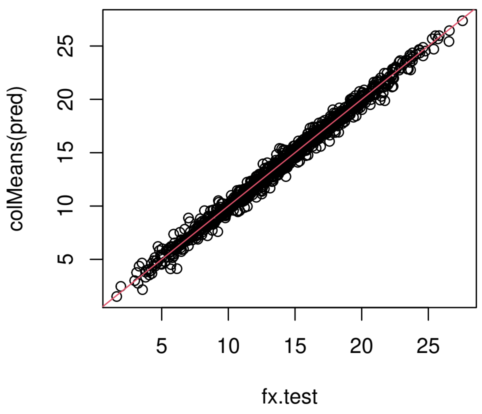

Figure 5: BASS prediction on test data – Friedman function.

```
Predict Start #-- May 07 16:40:27 --# Models: 592
Predict #-- May 07 16:40:27 --# Model: 100
Predict #-- May 07 16:40:27 --# Model: 200
Predict #-- May 07 16:40:27 --# Model: 300
Predict #-- May 07 16:40:27 --# Model: 400
Predict #-- May 07 16:40:27 --# Model: 500
```

Plotting these samples against the true values of $f(x)$ shows that we have a good fit (Figure 5).

```
R> fx.test <- f(x.test)
R> plot(fx.test, colMeans(pred))
R> abline(a = 0, b = 1, col = 2)
```

Now that we are considering a function of many variables, we may be interested in sensitivity analysis. To get the Sobol' decomposition for each posterior sample, we use the `sobol` function.

```
R> sens <- sobol(mod, verbose = FALSE)
```

Note that when `verbose = TRUE`, this function prints after every 10 models (as with the `predict` function, vectorizing around models rather than MCMC iterations saves a large amount of time). Depending on the number of basis functions and the number of models, this function can take significant amounts of time. If that is the case, using a smaller set of MCMC iterations by specifying `mcmc.use` may be useful.

The default plotting for this kind of object (Figure 6) shows boxplots of variance explained for each main effect and interaction that shows up in the BASS model. It also shows boxplots of the total sensitivity indices.

```
R> plot(sens, cex.axis = 0.5)
```

If there are a large number of main effects or interactions that explain very small percentages of variation, we can show only the effects that are most significant. For instance, we could show only the effects that, on average, explain at least 1% of the variance (Figure 7).

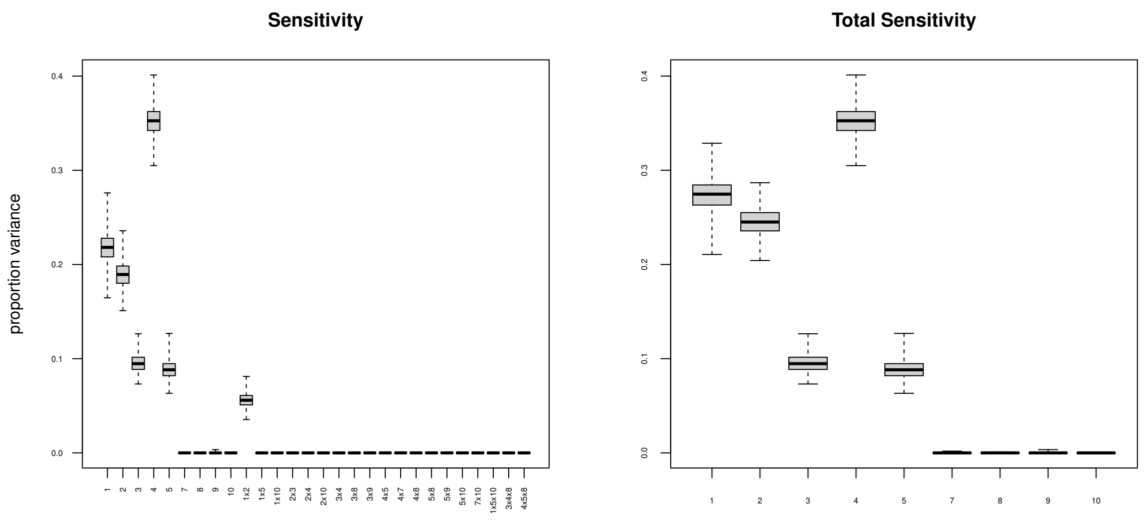

Figure 6: BASS sensitivity analysis – Friedman function.

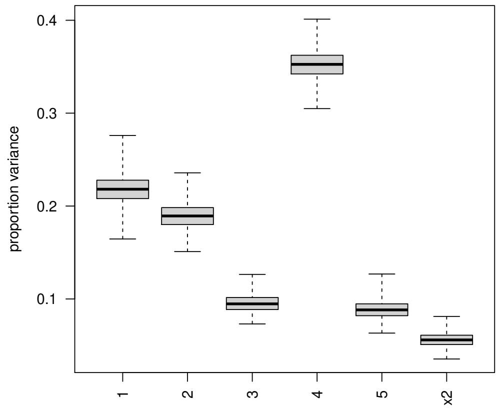

Figure 7: Most important main effects and interactions – Friedman function.

```
R> boxplot(sens$S[, colMeans(sens$S) > 0.01], las = 2,
+    ylab = "proportion variance", range = 0)
```

As expected, we see that almost all of the variance is from the first five variables and the only strong interaction is between the first two variables.

As a final note for this example, we discuss tempering diagnostics. We would like for neighboring chains to have swap acceptance rate of somewhere around 23%. Running `bass` with `verbose = TRUE` prints these acceptance rates every 1000 iterations. At the completion of the sampling, we can investigate acceptance rates by dividing the swap counts by the number of swap proposals, as follows.

```
R> mod$count.swap/mod$count.swap.prop

[1] 0.4703961 0.3873609 0.4641745 0.4766776 0.2188010 0.4721832 0.3771823
[8] 0.2198053
```

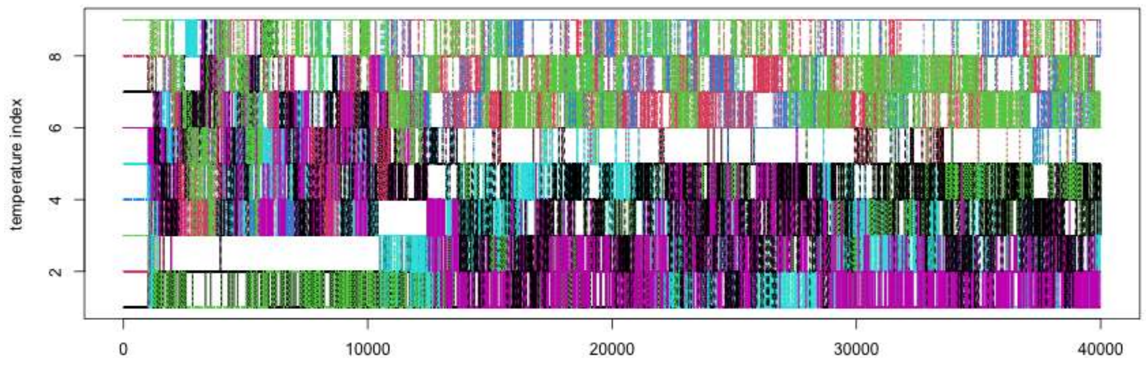

Figure 8: Parallel tempering diagnostics – swap trace plot.

Since we have specified nine temperatures, there are eight possible swaps, hence the eight numbers. If, for example, we wanted to increase the first acceptance rate, we would move the second temperature closer to the first.

Further analysis of swapping can be done by looking at swap trace plots.

```
R> matplot(mod$temp.val, type = "l", ylab = "temperature index")
```

Figure 8 shows the swap trace plot where the $y$-axis values are temperature indices (1 is the true posterior and 9 is the posterior raised to the smallest power), the $x$-axis shows MCMC iteration and the colored lines represent the different chains. We want to see these chains mixing throughout.

Determining whether the smallest value of the temperature ladder is small enough to allow for good mixing can be difficult. In this example, we could run the model with `temp.ladder = 11.03` and look at mixing diagnostics. One could also look at predicted versus observed plots at the different temperatures for the last MCMC iteration by executing the following code, the output of which is shown in Figure 9.

```
R> par(mfrow = c(3, 3))
R> temp.ind <- sapply(mod$curr.list, function(x) x$temp.ind)
R> for (i in 1:length(mod$temp.ladder)) {
+    ind <- which(temp.ind == i)
+    yhat <- mod$curr.list[[ind]]$des.basis %*% mod$curr.list[[ind]]$beta
+    plot(yhat, y, main = round(mod$temp.ladder[i], 2))
+    abline(a = 0, b = 1, col = 2)
+ }
```

Note that the `curr.list` object is a list with number of elements equal to the number of temperatures. This list contains the MCMC state for each chain. Since we swap temperatures rather than entire states, the chains are not in order according to temperature. We note that using the default prior for $\sigma^2$ with a temperature ladder with relatively large values can lead to instabilities when estimating $\sigma^2$. In cases where that is clearly the case, the prior for $\sigma^2$ will be automatically changed and a warning will be generated.

To demonstrate what is different when we use tempering, consider the equivalent BASS model fit without tempering.

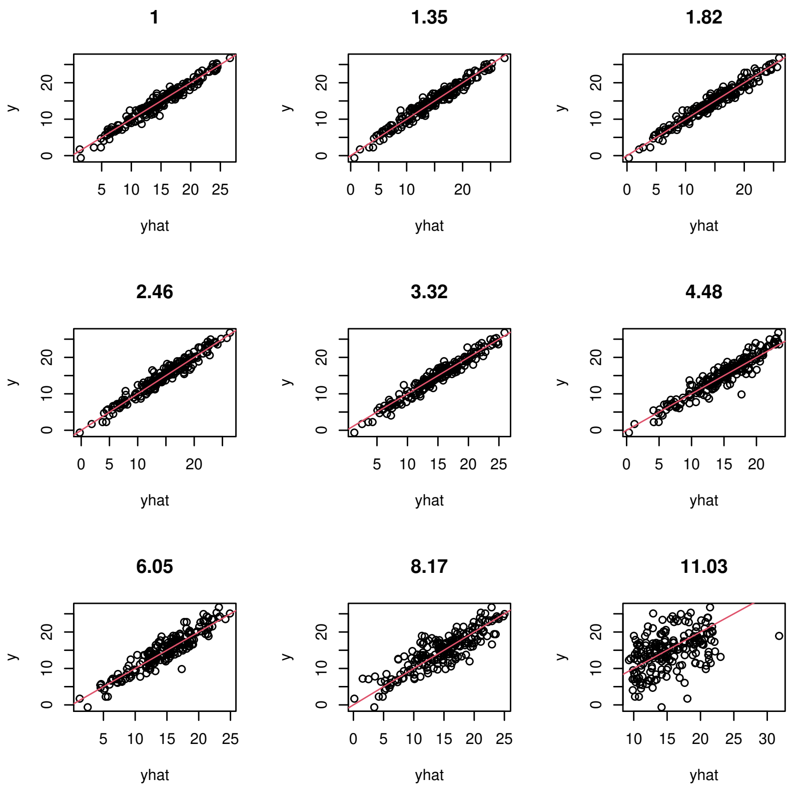

Figure 9: Predicted versus observed for the last MCMC iteration of the nine chains at different temperatures. The temperatures are shown above each plot.

```
R> mod.noTemp <- bass(x, y, nmcmc = 40000, nburn = 30000, thin = 10,
+    verbose = FALSE)
```

We compare the root mean square prediction error (RMSE) for the two models, as well as the empirical coverage of 95% probability intervals. First, the RMSE for the model fit without tempering

```
R> pred.noTemp <- predict(mod.noTemp, x.test)
R> sqrt(mean((colMeans(pred.noTemp) - fx.test)^2))

[1] 0.4994824
```

and the empirical coverage

```
R> quants.noTemp <- apply(pred.noTemp, 2, quantile, probs = c(0.025, 0.975))
R> mean((quants.noTemp[1, ] < fx.test) & (quants.noTemp[2, ] > fx.test))

[1] 0.887
```

demonstrate that the fit is quite good. When we use parallel tempering, the RMSE

```
R> sqrt(mean((colMeans(pred) - fx.test)^2))
```

```
[1] 0.4825888
```

and the empirical coverage

```
R> quants <- apply(pred, 2, quantile, probs = c(0.025, 0.975))
R> mean((quants[1, ] < fx.test) & (quants[2, ] > fx.test))
```

```
[1] 0.915
```

tend to be moderately better. Under different seeds, we tend to see higher coverage when we use tempering and lower coverage when we do not. We also tend to get better models in terms of RMSE when we use tempering. Other benefits of tempering will be shown in later examples. Because the computational burden is currently linear in the number of temperatures, using fewer temperatures is better. Thus, for many purposes, the model without tempering may be good enough.

We also point out that this modeling framework can handle correlated inputs. For instance, we use the correlation matrix

```
R> S <- matrix(0.99, nrow = 10, ncol = 10) + diag(10) * 0.01
```

as a covariance matrix for simulated inputs, and rescale them to be between zero and one so that Friedman function evaluations are comparable to the ones we have used above. The inputs are all highly correlated. The model fitting and prediction are done without any changes from what we have done previously.

```
R> library("MASS")
R> x <- mvrnorm(n, rep(0, 10), S)
R> x <- apply(x, 2, BASS:::scale.range)
R> y <- rnorm(n, f(x), sigma)
R> mod <- bass(x, y, nmcmc = 40000, nburn = 30000, thin = 10,
+    temp.ladder = (1 + 0.35)^(1:9 - 1), verbose = FALSE)
R> n.test <- 1000
R> x.test <- mvrnorm(n.test, rep(0, 10), S)
R> x.test <- apply(x.test, 2, BASS:::scale.range)
R> pred <- predict(mod, x.test)
```

We still get good prediction, as seen in Figure 10. We abstain from performing a sensitivity analysis under correlated input assumptions. Note that if we want to assume independence between the inputs when we do a sensitivity analysis, we can use correlated training data. However, we will likely be requiring the model to extrapolate outside the training data when we integrate over independent ranges, which can lead to poor results.

## 4.3. Friedman function with a categorical variable

In this example, we use data generated from a function similar to the Friedman function in the previous example but with a categorical variable included. The function, introduced in

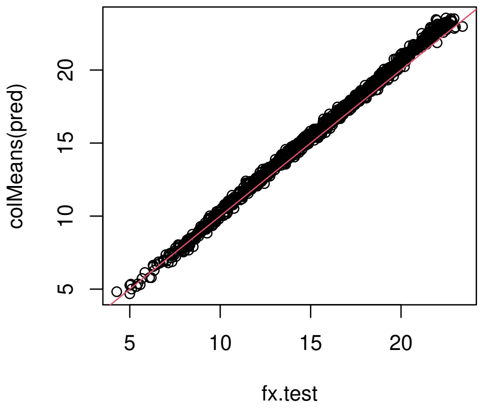

Figure 10: BASS prediction on test data – Friedman function with correlated inputs.

Gramacy and Taddy (2010), has

$$f(\mathbf{x}) = \begin{cases} 10\sin(\pi x_1 x_2) & x_{11} = 1 \\ 20(x_3 - 0.5)^2 & x_{11} = 2 \\ 10x_4 + 5x_5 & x_{11} = 3 \\ 5x_1 + 10x_2 + 20(x_3 - 0.5)^2 + 10\sin(\pi x_4 x_5) & x_{11} = 4 \end{cases}$$

as the mean function and standard Normal error. Again, $x_6, \ldots, x_{10}$ are unimportant. We generate 500 random uniform samples of the first 10 variables and randomly sample 500 values of the four categories of the 11th variable. The `bass` function treats input variables as categorical only if they are coded as factors.

```
R> set.seed(0)
R> f <- function(x) {
+    as.numeric(x[, 11] == 1) * (10 * sin(pi * x[, 1] * x[, 2])) +
+      as.numeric(x[, 11] == 2) * (20 * (x[, 3] - 0.5)^2) +
+        as.numeric(x[, 11] == 3) * (10 * x[, 4] + 5 * x[, 5]) +
+          as.numeric(x[, 11] == 4) * (10 * sin(pi * x[, 5] * x[, 4]) +
+            20 * (x[, 3] - 0.5)^2 + 10 * x[, 2] + 5 * x[, 1])
+ }
R> sigma <- 1
R> n <- 500
R> x <- data.frame(matrix(runif(n * 10), n, 10),
+    as.factor(sample(1:4, size = n, replace = TRUE)))
R> y <- rnorm(n, f(x), sigma)
```

We fit a model with tempering and use it for prediction, as in the previous example.

```
R> mod <- bass(x, y, nmcmc = 40000, nburn = 30000, thin = 10,
+    temp.ladder = (1 + 0.25)^(1:9 - 1), verbose = FALSE)
R> n.test <- 1000
```

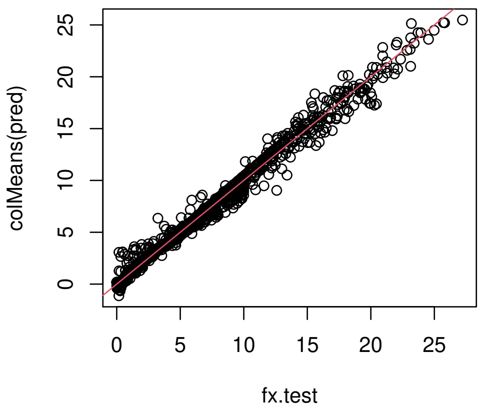

Figure 11: BASS prediction on test data – Friedman function with categorical predictor.

```
R> x.test <- data.frame(matrix(runif(n.test * 10), n.test, 10),
+    as.factor(sample(1:4, size = n.test, replace = TRUE)))
R> pred <- predict(mod, x.test)
```

Plotting posterior predictive samples against true values of $f(x)$ shows that we have a good fit (Figure 11).

```
R> fx.test <- f(x.test)
R> plot(fx.test, colMeans(pred))
R> abline(a = 0, b = 1, col = 2)
```

Sensitivity analysis is performed in the same manner.

```
R> sens <- sobol(mod)
```

Plotting the posterior distributions of the most important (explaining more than 0.5% of the variance) sensitivity indices in Figure 12, we see how important the categorical variable is as well as which variables it interacts with.

```
R> boxplot(sens$S[, colMeans(sens$S) > 0.005], las = 2,
+    ylab = "proportion variance", range = 0)
```

## 4.4. Friedman function with functional response

Next, we consider an extension of the Friedman function that is functional in one variable Francom *et al.* (2018). We use

$$f(\mathbf{x}) = 10\sin(2\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5$$

where we treat $x_1$ as the functional variable. Note that we insert a two into the `sin` function in order to increase the variability due to $x_1$ and $x_2$, making the problem more challenging. We generate 500 combinations of $x_2, \ldots, x_{10}$ from a uniform hypercube. We generate a grid of values of $x_1$ of length 50. This ends up being $500 \times 50$ combinations of inputs, for which we

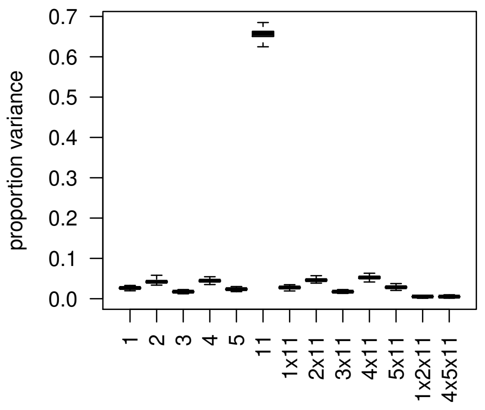

Figure 12: Most important main effects and interactions – Friedman function with categorical predictor.

evaluate $f$ and add standard Normal error. We keep the responses in a matrix of dimension $500 \times 50$ so that each row represents a curve. The inputs are kept separate in a $500 \times 9$ matrix and a grid of length 50.

```
R> set.seed(0)
R> f <- function(x) {
+    10 * sin(2 * pi * x[, 1] * x[, 2]) + 20 * (x[, 3] - 0.5)^2 +
+      10 * x[, 4] + 5 * x[, 5]
+ }
R> sigma <- 1
R> n <- 500
R> n.func <- 50
R> x.func <- seq(0, 1, length.out = n.func)
R> x <- matrix(runif(n * 9), n)
R> y <- matrix(f(cbind(rep(x.func, each = n),
+    kronecker(rep(1, n.func), x))), ncol = n.func) +
+      rnorm(n * n.func, 0, sigma)
```

The functional data can be plotted as follows and are shown in Figure 13.

```
R> matplot(x.func, t(y), type = "l")
```

In order for the **BASS** package to handle functional responses, each curve needs to be evaluated on the same grid. Thus, the responses must be able to be stored as a matrix without missing values.

### *Augmentation approach*

We fit the augmentation functional response model by specifying our matrices `x` and `y` as well as the grid `x.func`.

```
R> mod <- bass(x, y, xx.func = x.func)
```

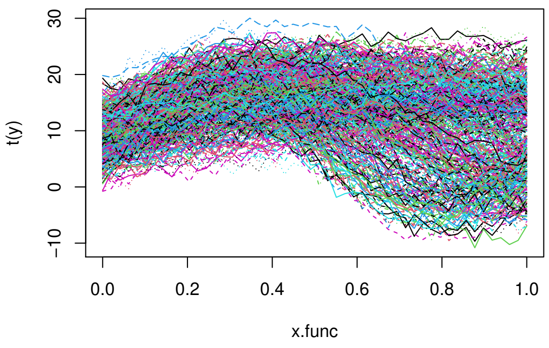

Figure 13: 500 functional responses. The goal is to fit a functional nonparametric regression model and perform sensitivity analysis.

```
MCMC Start #-- May 07 16:49:26 --# nbasis: 0
MCMC iteration 1000 #-- May 07 16:49:28 --# nbasis: 67
MCMC iteration 2000 #-- May 07 16:49:30 --# nbasis: 65
MCMC iteration 3000 #-- May 07 16:49:33 --# nbasis: 46
MCMC iteration 4000 #-- May 07 16:49:34 --# nbasis: 46
MCMC iteration 5000 #-- May 07 16:49:35 --# nbasis: 38
MCMC iteration 6000 #-- May 07 16:49:36 --# nbasis: 38
MCMC iteration 7000 #-- May 07 16:49:37 --# nbasis: 39
MCMC iteration 8000 #-- May 07 16:49:38 --# nbasis: 38
MCMC iteration 9000 #-- May 07 16:49:39 --# nbasis: 38
MCMC iteration 10000 #-- May 07 16:49:42 --# nbasis: 38
```

Prediction is as simple as before. If we want to predict on a different functional grid, we can specify that in the `predict` function with `newdata.func`.

```
R> n.test <- 100
R> x.test <- matrix(runif(n.test * 9), n.test)
R> pred <- predict(mod, x.test)
```

Following, we make a functional predicted versus observed plot, shown in Figure 14.

```
R> fx.test <- matrix(f(cbind(rep(x.func, each = n.test),
+    kronecker(rep(1, n.func), x.test))), ncol = n.func)
R> matplot(fx.test, apply(pred, 2:3, mean), type = "l")
R> abline(a = 0, b = 1, col = 2)
```

We will demonstrate the two methods of sensitivity analysis discussed in Section 3. First, we can get the Sobol' indices for the functional variable and its interactions just as we do the other variables. This is the default.

```
R> sens <- sobol(mod, mcmc.use = 1:100)
```

```
Sobol Start #-- May 07 16:49:55 --# Models: 2
Total Sensitivity #-- May 07 16:49:56 --#
```

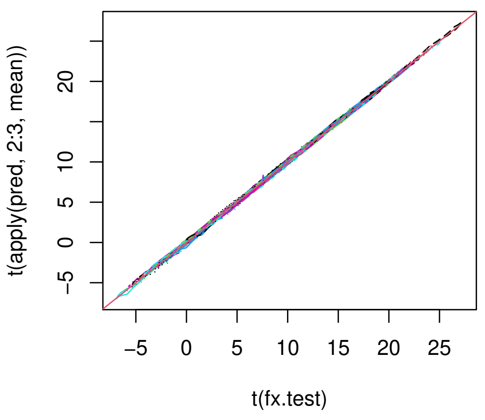

Figure 14: BASS prediction performance – Friedman function with functional response.

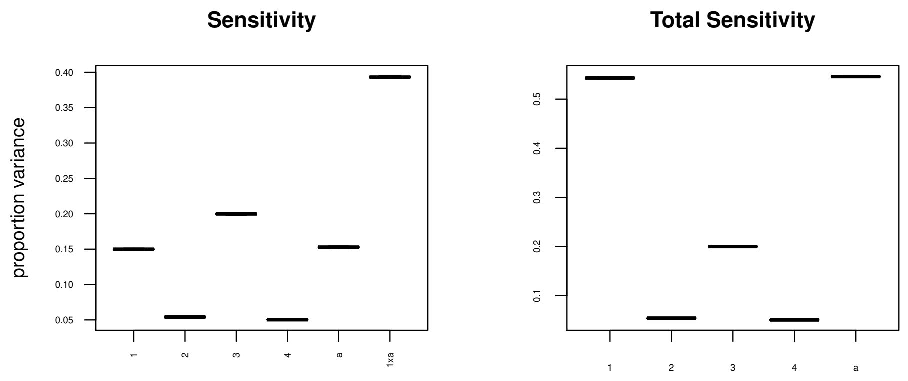

Figure 15: Sensitivity analysis – Friedman function with functional response.

When we plot the variance decomposition, as shown in Figure 15, the functional variable is labeled with the letter "a." If we had multiple functional variables, they would be labeled with different letters.

```
R> plot(sens, cex.axis = 0.5)
```

The other approach to sensitivity analysis is to get a functional variance decomposition. This is done by using the `func.var` parameter. If there is only one functional variable, we set `func.var = 1`. Otherwise we set `func.var` to the column of `xx.func` we want to use for our functional variance decomposition.

```
R> sens.func <- sobol(mod, mcmc.use = 1:100, func.var = 1)

Sobol Start #-- May 07 16:49:56 --# Models: 2
```

When we plot the variance decomposition, shown in Figure 16, we get two plots.

```
R> plot(sens.func)
```

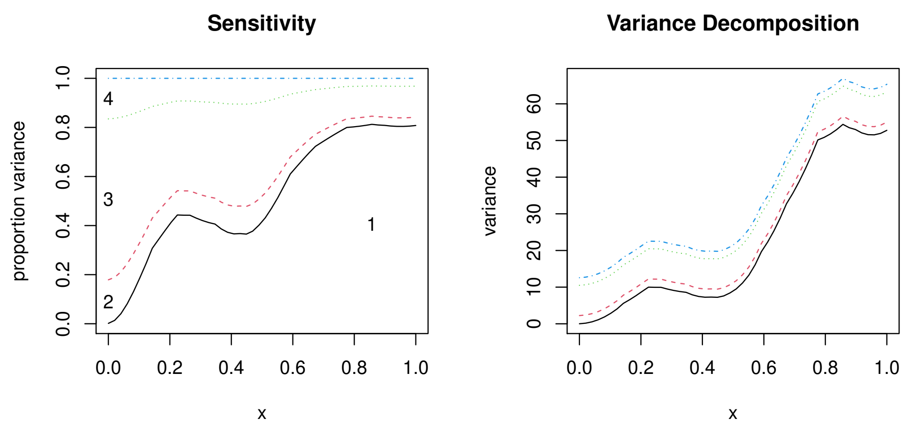

Figure 16: Functional sensitivity analysis – Friedman function with functional response.

The left plot shows the posterior mean (using posterior samples specified with `mcmc.use`) of the functional sensitivity indices in a functional pie chart. The right plot shows the variance decomposition as a function of the functional variable. Thus, the top line in the right plot is the total variance in $y$ as a function of $x_1$. The bottom line (black) is the total variance explained by the main effect of $x_2$ as a function of $x_1$. The labels in the plot on the left are the variable numbers (columns of `x`).

*Basis expansion approach*

We will utilize parallel computing in the following examples. The **BASS** functions that can utilize multiple threads can do so in two ways: fork or socket, specified with the `parType` option. We recommend the fork method, though that is not available on Windows. We would usually specify `n.cores = parallel::detectCores()` in these functions, but for the purposes of making this document compile on CRAN, we limit the number of threads used with the following specification.

```
R> if (.Platform$OS.type == "unix") {
+    nc <- 2
+  } else{
+    nc <- 1
+  }
```

We should also point out that this kind of explicit parallelism sometimes does not play well with multithreading that comes from **BLAS**. Functions in the **BASS** package benefit from both forms of parallelism, though we would prefer the explicit parallelism over the multithreading when possible (which seems to happen by default when using **openBLAS**).

The basis approach to functional response modeling using a principal component basis can be done as follows:

```
R> mod.pca <- bassPCA(x, y, perc.var = 95, n.cores = nc)
R> pred.pca <- predict(mod.pca, x.test)
R> sens.func.pca <- sobolBasis(mod.pca, int.order = 2,
+    mcmc.use = 100, n.cores = nc, verbose = FALSE)
```

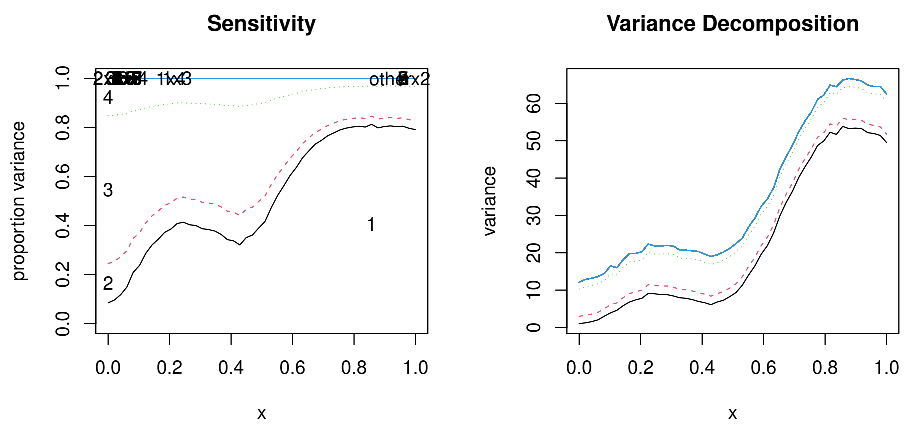

Figure 17: Functional sensitivity analysis, PCA space – Friedman function with functional response.

The optional `perc.var` argument specifies how many principal components should be used in terms of the percent of variance explained. We could alternately specify `n.pc`. Then `n.pc` BASS models are fit independently, using `n.cores` threads. To limit computation, the `sobolBasis` function requires a specified highest degree of interaction in the decomposition, `int.order`, and a single MCMC sample to use, `mcmc.use`. This function benefits from having many threads (often more so than `bassPCA`). Figure 17 shows the functional variance decomposition.

```
R> plot(sens.func.pca)
```

The `bassPCA` function is a shortcut to the `bassBasis` function when we want to use a principal component basis. For other bases, we use the `bassBasis` function directly. We demonstrate this with a wavelet basis below.

First, we must interpolate our functional response data onto a grid of power two.

```
R> pow2 <- floor(log2(ncol(y)))
R> y.pow2 <- matrix(nrow = n, ncol = 2^pow2)
R> for (i in 1:n) {
+    y.pow2[i,] <- approx(1:ncol(y), y[i, ],
+      xout = seq(1, ncol(y), length.out = 2^pow2))$y
+  }
```

Then we (optionally) subtract the mean function from each functional response.

```
R> y.pow2.mean <- colMeans(y.pow2)
R> y.pow2.scale <- sweep(y.pow2, 2, y.pow2.mean)
```

We use the **wmtsa** package (Constantine and Percival 2017) to perform the wavelet decomposition for each functional response.

```
R> library("wmtsa")
R> coefs.mat <- matrix(nrow = 2^pow2, ncol = n)
```

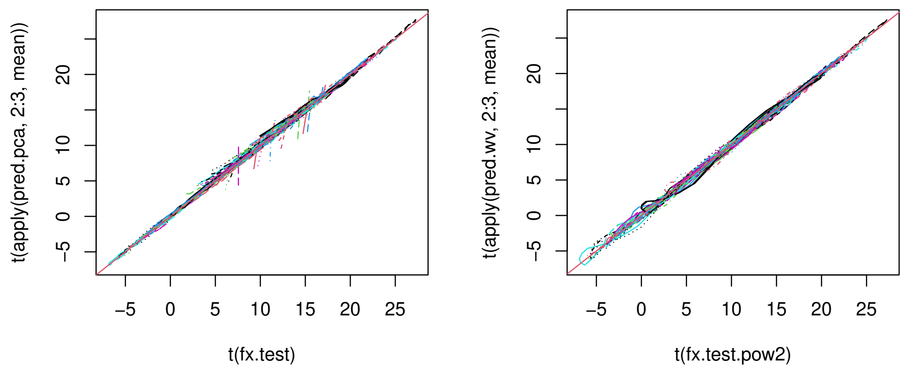

Figure 18: BASS prediction performance, PCA (left) and wavelet (right) space – Friedman function with functional response.

```
R> for (i in 1:nrow(y.pow2.scale)) {
+    w <- wavDWT(y.pow2.scale[i, ])
+    coefs.mat[, i] <- unlist(w$data)
+  }
```

We then threshold the wavelet coefficients at 5% of the largest magnitude wavelet coefficient.

```
R> thresh <- 0.05 * max(abs(coefs.mat))
R> use <- unique(which(coefs.mat > thresh, arr.ind = TRUE)[, 1])
```

Finally, we specify the matrix of basis functions and our reduced dimension response as part of the list that will be passed to the `bassBasis`.

```
R> basis <- t(wavDWTMatrix(J = pow2, wavelet = "s8"))[, use]
R> newy <- coefs.mat[use, ]
R> dat <- list(xx = x, y = y.pow2, n.pc = ncol(basis), basis = basis,
+    newy = newy, y.m = y.pow2.mean, y.s = rep(1, 2^pow2))
```

Then we can call the `bassBasis`, `predict`, and `sobolBasis` functions.

```
R> mod.wv <- bassBasis(dat, n.cores = nc)
R> pred.wv <- predict(mod.wv, x.test)
R> sens.func.wv <- sobolBasis(mod.wv, int.order = 2, mcmc.use = 100,
+    n.cores = nc, verbose = FALSE)
```

To compare predictions, we first interpolate our test data onto a grid of power two.

```
R> fx.test.pow2 <- matrix(nrow = n.test, ncol = 2^pow2)
R> for (i in 1:n.test) {
+    fx.test.pow2[i,] <- approx(1:ncol(fx.test), fx.test[i, ],
+      xout = seq(1, ncol(fx.test), length.out = 2^pow2))$y
+  }
```

The prediction accuracy plots for the PCA basis approach and the wavelet basis approach are shown in Figure 18.

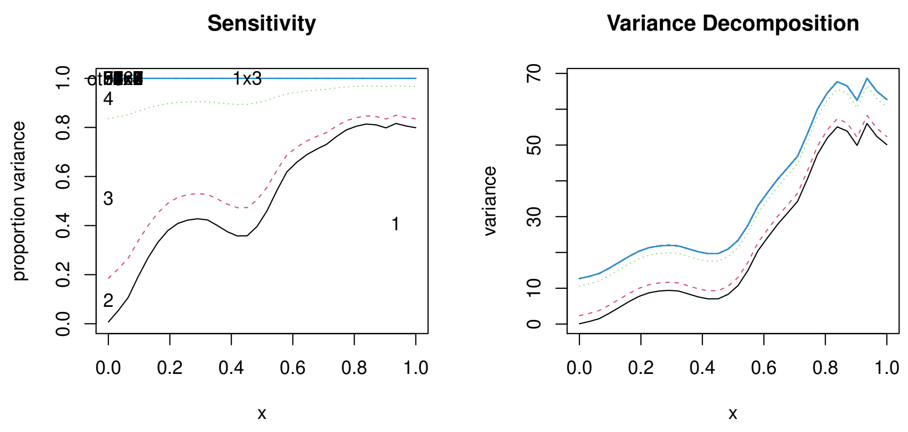

Figure 19: Functional sensitivity analysis, wavelet space – Friedman function with functional response.

```
R> par(mfrow = c(1, 2))
R> matplot(t(fx.test), t(apply(pred.pca, 2:3, mean)), type = "l")
R> abline(a = 0, b = 1, col = 2)
R> matplot(t(fx.test.pow2), t(apply(pred.wv, 2:3, mean)), type = "l")
R> abline(a = 0, b = 1, col = 2)
```

Finally, we also show the functional Sobol decomposition in Figure 19.

```
R> plot(sens.func.wv)
```

## 4.5. Air foil data

In this example, we consider a NASA data set, obtained from a series of aerodynamic and acoustic tests of two- and three-dimensional airfoil blade sections conducted in an anechoic wind tunnel (Lichman 2013). The response is scaled sound pressure level, in decibels. There are five inputs: (1) frequency, in Hertzs; (2) angle of attack, in degrees; (3) chord length, in meters; (4) free-stream velocity, in meters per second; and (5) suction side displacement thickness, in meters. The data have 1503 combinations of these inputs, some of which are collinear (variables 2 and 5 have correlation of 0.75).

```
R> dd <- read.table("airfoil_self_noise.dat")
```

We set aside 200 input combinations to use for testing.

```
R> set.seed(0)
R> test <- sample(nrow(dd), size = 150)
R> x <- dd[-test, 1:5]
R> y <- dd[-test, 6]
```

We fit a BASS model using tempering.

```
R> mod <- bass(x, y, nmcmc = 20000, nburn = 10000, thin = 10,
+    temp.ladder = 1.1^(0:5), verbose = FALSE)
```

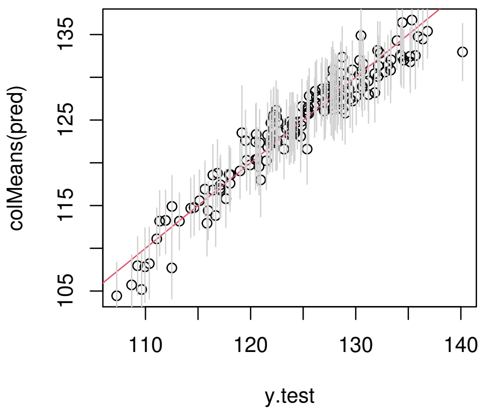

Figure 20: Prediction performance – air foil data.

We can predict as we have before. However, this prediction is for the mean function.

```
R> x.test <- dd[test, 1:5]
R> y.test <- dd[test, 6]
R> pred <- predict(mod, x.test)
```

Now, if we are interested in predicting actual data rather than the mean function, we can incorporate uncertainty from our estimate of $\sigma^2$.

```
R> pred.error <- pred + matrix(rnorm(nrow(pred) * ncol(pred), 0,
+    sqrt(mod$s2)), nrow = nrow(pred))
R> q1 <- apply(pred.error, 2, quantile, probs = 0.05)
R> q2 <- apply(pred.error, 2, quantile, probs = 0.95)
R> mean((q1 < y.test) & (q2 > y.test))
```

```
[1] 0.9333333
```

This puts our empirical coverage near where we would expect it to be. We can plot our 90% prediction intervals as follows, shown in Figure 20.

```
R> plot(y.test, colMeans(pred))
R> abline(a = 0, b = 1, col = 2)
R> segments(y.test, q1, y.test, q2, col = "lightgrey")
```

Next, we can obtain and plot the Sobol' decomposition, shown in Figure 21. We disregard the dependence among the input variables.

```
R> sens <- sobol(mod, verbose = FALSE)
R> plot(sens)
```

The uncertainty in the sensitivity indices in Figure 21 is significant and helps us to understand that there are many possible models for these data that use different variables and interactions. The proper characterization of this uncertainty would be impossible if our RJMCMC chain

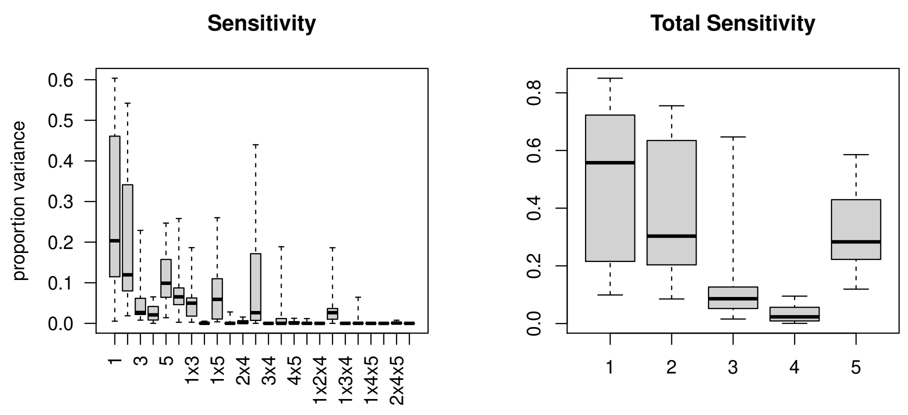

Figure 21: Sobol' decomposition – air foil data.

was stuck in a mode. Hence, tempering is important in this problem. By exploring the posterior modes, tempering allows us to find not just a model that predicts well, but all the models that predict well.

## 4.6. Pollutant spill model

The final example we present is an emulation problem. The simulator is for modeling a pollutant spill caused by a chemical accident, obtained from Surjanovic and Bingham (2017). While fast to evaluate, this simulator provides a good testbed for BASS methods. The simulator has four inputs: (1) mass of pollutant spilled at each of two locations (range 7–13), (2) diffusion rate in the channel (0.02–0.12), (3) location of the second spill (0.01–3), and (4) time of the second spill (30.01–30.295). The simulator outputs a function in space (one dimension) and time that is the concentration of the pollutant.

We generate 1000 combinations of the four simulator inputs uniformly from within their respective ranges.

```
R> set.seed(0)
R> n <- 1000
R> x <- cbind(runif(n, 7, 13), runif(n, 0.02, 0.12), runif(n, 0.01, 3),
+    runif(n, 30.01, 30.295))
```

We specify six points in space and 20 time points. The functional grid that would be passed to the function `bass` would thus have two columns, called `x.func` below.

```
R> s <- c(0, 0.5, 1, 1.5, 2, 2.5)
R> t <- seq(0.3, 60, length.out = 20)
R> x.func <- expand.grid(t, s)
```

We will show results when using the `bassPCA` function rather than the `bass` function for this problem. The reader may wish to test different parameter ranges, numbers of simulations, and functional response modeling approaches. Our experience is that the `bassPCA` function performs quite well for this problem, and that various BASS models can handle tens of thousands of model runs.

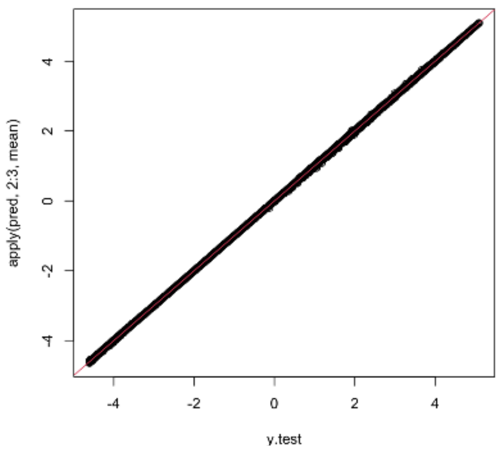

Figure 22: BASS prediction performance – pollutant spill model.

We use function `environ` available from `http://www.sfu.ca/~ssurjano/Code/environr.html` to generate realizations of the simulator. We will model the log of the simulator output, though plume models like this may deserve better thought out transformations, as in Bliznyuk, Ruppert, Shoemaker, Regis, Wild, and Mugunthan (2008).

```
R> out <- t(apply(x, 1, environ, s = s, t = t))
R> y <- log(out + 0.01)
```

The model is fit as follows.

```
R> mod <- bassPCA(x, y, n.pc = 20, save.yhat = FALSE,
+    n.cores = nc, verbose = FALSE)
```

Note that we specify `save.yhat = FALSE`. By default, the `bass` function saves in-sample predictions for all MCMC samples (post burn-in and thinned). This can be a significant storage burden when we have large amounts of functional data, as we do in this case. Changing the `save.yhat` parameter can relieve this. If in-sample predictions are of interest, they can be obtained after model fitting using the `predict` function.

As with the previous example, prediction here is for the mean function. Whatever error is left over (in $\sigma^2$) is inability of the BASS model to pick up high frequency signal.

```
R> n.test <- 1000
R> x.test <- cbind(runif(n.test, 7, 13), runif(n.test, 0.02, 0.12),
+    runif(n.test, 0.01, 3), runif(n.test, 30.01, 30.295))
R> y.test <- log(t(apply(x.test, 1, environ, s = s, t = t)) + 0.01)
R> pred <- predict(mod, x.test)
```

A plot of the predicted (mean function) versus observed data is shown in Figure 22.

```
R> plot(y.test, apply(pred, 2:3, mean))
R> abline(a = 0, b = 1, col = 2)
```

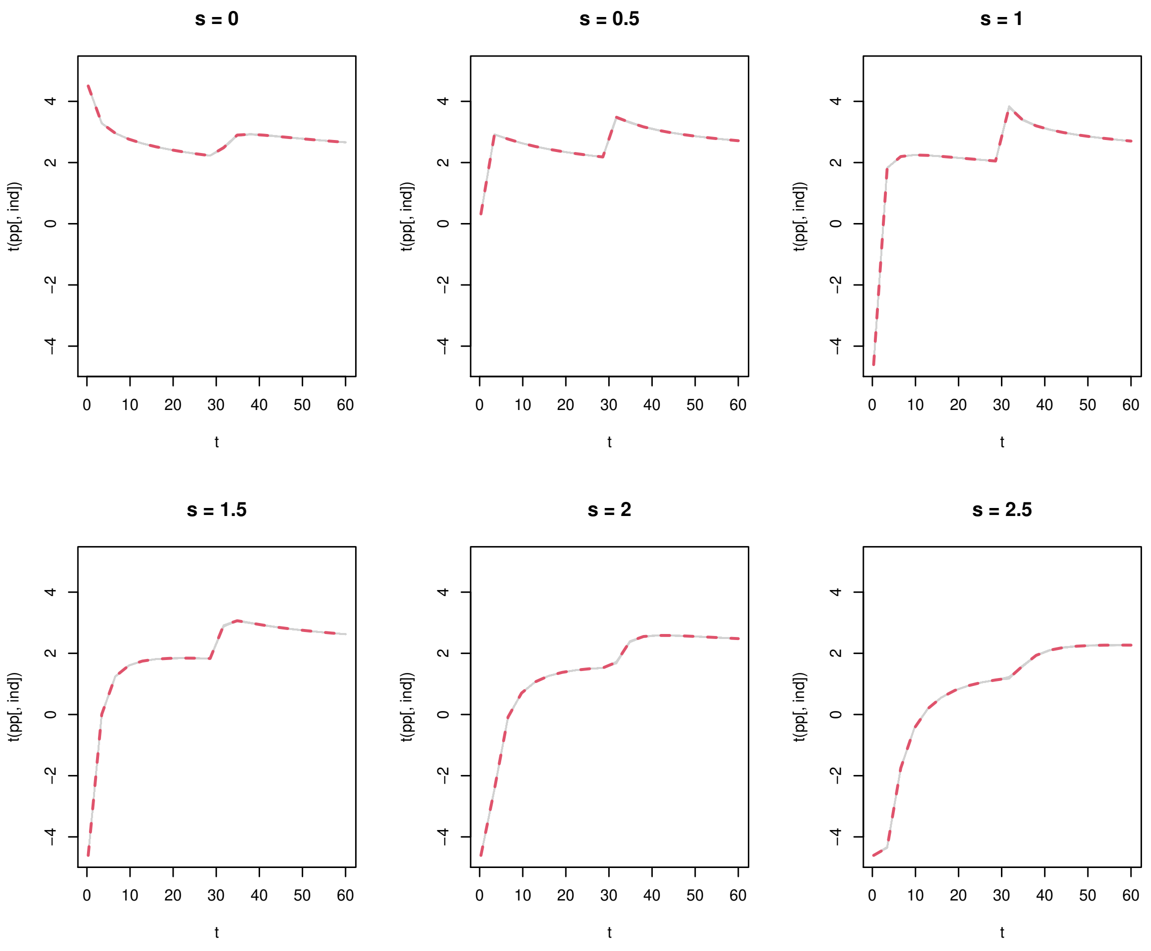

Figure 23: BASS prediction in space and time – pollutant spill model.

To see what the predictions look like in space and time, consider the plots shown in Figure 23. These show posterior draws (in gray) of the mean function for one setting of the four inputs along with simulator output (in red).

```
R> pp <- pred[, 1, ]
R> ylim <- range(y)
R> par(mfrow = c(2, 3))
R> for (i in 1:length(s)) {
+    ind <- length(t) * (i - 1) + 1:length(t)
+    matplot(t, t(pp[, ind]), type = "l", col = "lightgrey",
+      ylim = ylim, main = paste("s =", s[i]))
+    lines(t, y.test[1, ind], col = 2, lwd = 2, lty = 2)
+  }
```

Below, we show how to get spatio-temporal Sobol' indices. We limit the models considered using `mcmc.use` to speed up computations.

```
R> sens.func <- sobolBasis(mod, mcmc.use = 1, int.order = 2, verbose = FALSE,
+    n.cores = nc)
```

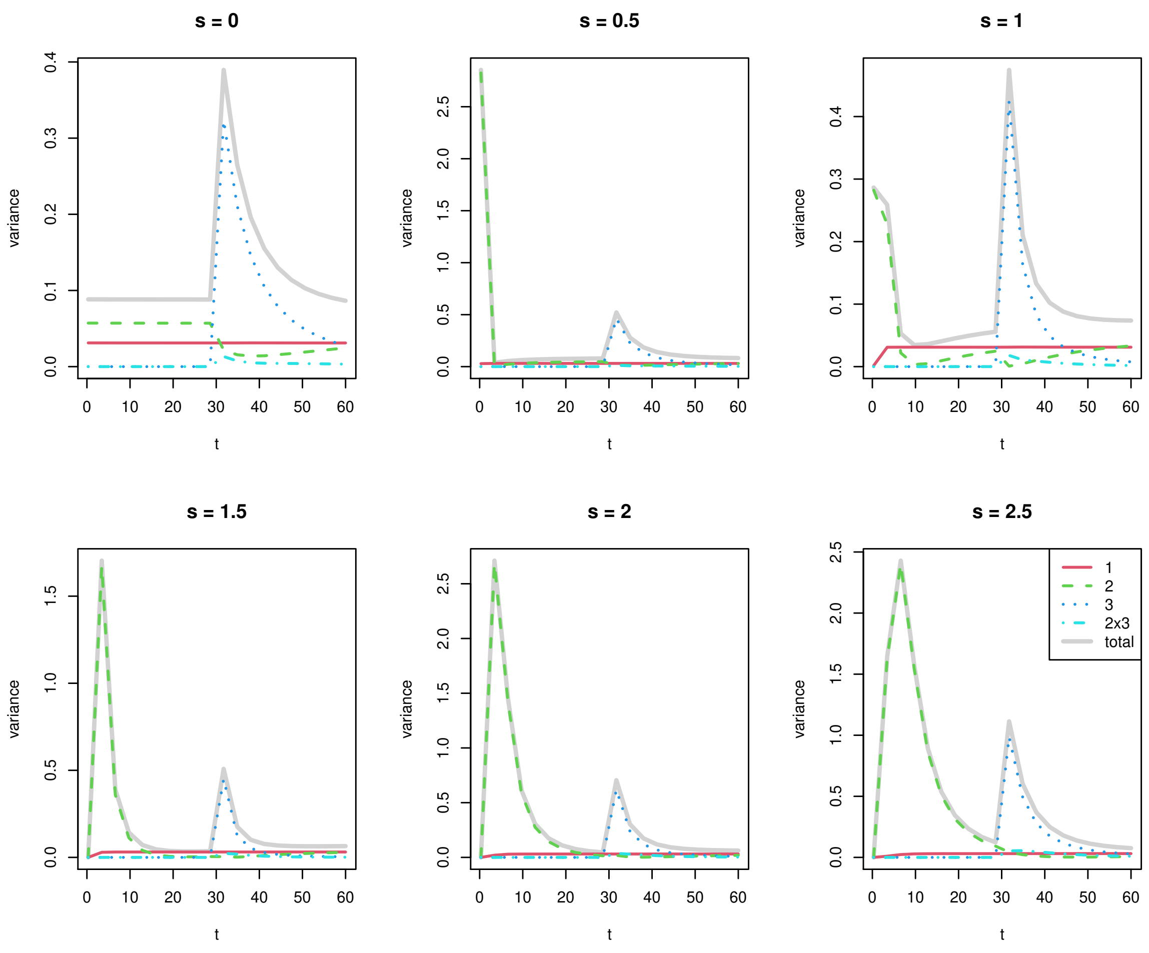

Figure 24: Variance decomposition as a function of space and time – pollutant spill model.

To show plots of the Sobol' indices over space and time, we can use a similar approach to what we used to show predictions, above. Here, we limit the effects shown by thresholding their integrated variance, where we use 1% of the integrated total variance as the cutoff.

```
R> use <- which(apply(sens.func$S.var, 2, mean) >
+    mean(sens.func$Var.tot) * 0.01)
R> par(mfrow = c(2, 3))
R> for (i in 1:length(s)) {
+    ind <- length(t) * (i - 1) + 1:length(t)
+    plot(t, sens.func$Var.tot[ind, 1], type = "l", lwd = 3,
+      ylim = c(0, max(sens.func$Var.tot[ind, 1])), ylab = "variance",
+      col = "lightgrey", main = paste("s =", s[i]))
+    matplot(t, t(sens.func$S.var[1, use, ind]), type = "l",
+      main = paste("s =", s[i]), add = TRUE, col = 2:5, lwd = 2)
+ }
R> legend("topright", c(sens.func$names.ind[use], "total"),
+    col = c(2:5, "lightgrey"), lty = c(1:4, 1), lwd = c(rep(2, 4), 3))
```

These plots demonstrate that, for most spatial locations, the time series of variance has two peaks for the two different spills. Within the range we have selected, the first input, pollutant mass, has fairly uniform influence across the time series. The second input, the diffusion rate in the channel has a very large influence on the concentration of the pollutant during the first spill, while the third input, the location of the second spill, has the largest effect on the concentration during the second spill. These two inputs have a small interaction effect that is most prominent at the beginning of the second spill. Notably, the fourth parameter, the time of the second spill, has no effect, likely because the range used for that parameter was very small.

## 5. Summary

Our proposed BASS framework provides a powerful general tool for nonparametric regression settings. It can be used for modeling with many continuous and categorical inputs, large sample size and functional response. It provides posterior sensitivity analyses without integration error. The MCMC approach to inference, especially using parallel tempering, yields posterior samples that can be used for probabilistic prediction. The **BASS** package makes these features accessible to users with minimal exposure. These capabilities have been demonstrated with a set of examples involving different dimensions, categorical variables, functional responses, and large datasets.

## References

Altekar G, Dwarkadas S, Huelsenbeck JP, Ronquist F (2004). "Parallel Metropolis Coupled Markov Chain Monte Carlo for Bayesian Phylogenetic Inference." *Bioinformatics*, **20**(3), 407–415. doi:10.1093/bioinformatics/btg427.

Belitz C, Brezger A, Kneib T, Lang S, Umlauf N (2017). ***BayesX**: Software for Bayesian Inference in Structured Additive Regression Models.* Version 1.1, URL http://www.BayesX.org/.

Bliznyuk N, Ruppert D, Shoemaker C, Regis R, Wild S, Mugunthan P (2008). "Bayesian Calibration and Uncertainty Analysis for Computationally Expensive Models Using Optimization and Radial Basis Function Approximation." *Journal of Computational and Graphical Statistics*, **17**(2), 270–294. doi:10.1198/106186008x320681.

Constantine W, Percival D (2017). ***wmtsa**: Wavelet Methods for Time Series Analysis.* R package version 2.0-3, URL https://CRAN.R-project.org/package=wmtsa.

Denison DGT, Holmes CCC, Mallick BK, Smith AFM (2002). *Bayesian Methods for Nonlinear Classification and Regression*, volume 386. John Wiley & Sons.

Denison DGT, Mallick BK, Smith AFM (1998). "Bayesian MARS." *Statistics and Computing*, **8**(4), 337–346. doi:10.1023/a:1008824606259.

Fog A (2015). ***BiasedUrn**: Biased Urn Model Distributions.* R package version 1.07, URL https://CRAN.R-project.org/package=BiasedUrn.

Francom D (2020). ***BASS**: Bayesian Adaptive Spline Surfaces.* R package version 1.2.2, URL `https://CRAN.R-project.org/package=BASS`.

Francom D, Sansó B, Bulaevskaya V, Lucas D, Simpson M (2019). "Inferring Atmospheric Release Characteristics in a Large Computer Experiment using Bayesian Adaptive Splines." *Journal of the American Statistical Association*, **114**(528), 1450–1465. `doi:10.1080/01621459.2018.1562933`.

Francom D, Sansó B, Kupresanin A, Johannesson G (2018). "Sensitivity Analysis and Emulation for Functional Data Using Bayesian Adaptive Splines." *Statistica Sinica*, **28**(2), 791–816. `doi:10.5705/ss.202016.0130`.

Friedman JH (1991a). "Estimating Functions of Mixed Ordinal and Categorical Variables Using Adaptive Splines." *Technical report*, DTIC Document.

Friedman JH (1991b). "Multivariate Adaptive Regression Splines." *The Annals of Statistics*, **19**(1), 1–67. `doi:10.1214/aos/1176347963`.

Gramacy R, Samworth R, King R (2010). "Importance Tempering." *Statistics and Computing*, **20**(1), 1–7. `doi:10.1007/s11222-008-9108-5`.

Gramacy RB, Taddy M (2010). "Categorical Inputs, Sensitivity Analysis, Optimization and Importance Tempering with **tgp** Version 2, an R Package for Treed Gaussian Process Models." *Journal of Statistical Software*, **33**(6), 1–48. `doi:10.18637/jss.v033.i06`.

Green PJ (1995). "Reversible Jump Markov Chain Monte Carlo Computation and Bayesian Model Determination." *Biometrika*, **82**(4), 711–732. `doi:10.1093/biomet/82.4.711`.

Gu C (2014). "Smoothing Spline ANOVA Models: R Package **gss**." *Journal of Statistical Software*, **58**(5), 1–25. `doi:10.18637/jss.v058.i05`.

Hastie T, Tibshirani R (2017). ***mda**: Mixture and Flexible Discriminant Analysis.* R package version 0.4-10, URL `https://CRAN.R-project.org/package=mda`.

Kooperberg C (2019). ***polspline**: Polynomial Spline Routines.* R package version 1.1.16, URL `https://CRAN.R-project.org/package=polspline`.

Liang F, Paulo R, Molina G, Clyde MA, Berger JO (2008). "Mixtures of $g$ Priors for Bayesian Variable Selection." *Journal of the American Statistical Association*, **103**(481), 410–423. `doi:10.1198/016214507000001337`.

Lichman M (2013). "UCI Machine Learning Repository." URL `http://archive.ics.uci.edu/ml`.

McCulloch R, Sparapani R, Gramacy R, Spanbauer C, Pratola M (2019). ***BART**: Bayesian Additive Regression Trees.* R package version 2.7, URL `https://CRAN.R-project.org/package=BART`.

Milborrow S (2019). ***earth**: Multivariate Adaptive Regression Splines.* R package version 5.1.1, URL `https://CRAN.R-project.org/package=earth`.

Nott DJ, Kuk AYC, Duc H (2005). "Efficient Sampling Schemes for Bayesian MARS Models with Many Predictors." *Statistics and Computing*, **15**(2), 93–101. doi:10.1007/s11222-005-6201-x.

Racine JS, Nie Z (2018). ***crs**: Categorical Regression Splines.* R package version 0.15-31, URL https://CRAN.R-project.org/package=crs.

R Core Team (2020). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Saltelli A, Ratto M, Andres T, Campolongo F, Cariboni J, Gatelli D, Saisana M, Tarantola S (2008). *Global Sensitivity Analysis: The Primer.* John Wiley & Sons.

Sobol' IM (2001). "Global Sensitivity Indices for Nonlinear Mathematical Models and Their Monte Carlo Estimates." *Mathematics and Computers in Simulation*, **55**(1–3), 271–280. doi:10.1016/s0378-4754(00)00270-6.

Surjanovic S, Bingham D (2017). "Virtual Library of Simulation Experiments: Test Functions and Datasets." Retrieved January 5, 2017, from http://www.sfu.ca/~ssurjano.

The MathWorks Inc (2019). *MATLAB – The Language of Technical Computing, Version R2019a.* Natick. URL http://www.mathworks.com/products/matlab/.

Umlauf N, Adler D, Kneib T, Lang S, Zeileis A (2015). "Structured Additive Regression Models: An R Interface to **BayesX**." *Journal of Statistical Software*, **63**(21), 1–46. URL http://www.jstatsoft.org/v63/i21/.

Wood SN (2017). *Generalized Additive Models: An Introduction with R.* 2nd edition. Chapman & Hall/CRC, Boca Raton.

Wood SN (2019). ***mgcv**: Mixed GAM Computation Vehicle with Automatic Smoothness Estimation.* R package version 1.8-30, URL https://CRAN.R-project.org/package=mgcv.

Xie Y (2015). *Dynamic Documents with R and **knitr**.* 2nd edition. Chapman & Hall/CRC, Boca Raton. URL https://yihui.name/knitr/.

## Affiliation:

Devin Francom
Statistical Sciences Group (CCS-6)
Los Alamos National Laboratory
Los Alamos, NM 87545, United States of America
E-mail: dfrancom@lanl.gov
LA-UR-18-21548

*Journal of Statistical Software* http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
August 2020, Volume 94, Issue 8 *Submitted:* 2017-01-10
doi:10.18637/jss.v094.i08 *Accepted:* 2018-08-12