Bayesian treed Gaussian process models: A guide to the tgp package, v1.0

Robert B. Gramacy
Department of Applied Math & Statistics
University of California, Santa Cruz
rbgramacy@ams.ucsc.edu

December 23, 2005

The `tgp` package for `R` [17] is a tool for fully Bayesian, nonparametric, semiparametric, and nonstationary regression by treed Gaussian processes with jumps to the limiting linear model. Special cases also implemented include Bayesian linear models, linear CART, stationary separable and isotropic Gaussian processes. In addition to inference and posterior prediction, 1-d and 2-d plotting, with higher dimension projection and slice capabilities, and tree drawing functions are also provided for visualization of `tgp`-class output. (2-d plotting requires the `akima` library; tree plotting requires the `maptree` and `combinat` libraries.)

This document is intended to familiarize a (potential) user of `tgp` with the models and analyses available in the package. After a brief overview, the brunt of this document consists of examples, on mainly synthetic and randomly generated data, which illustrate the various functions and methodologies implemented by the package. This document has been authored in `Sweave` (try `help(Sweave)`). This means that (1) the code quoted throughout is certified by `R`, and the `Stangle` command can be used to extract it; and (2) that this is a dynamic document, i.e., each time the document is compiled, the figures and analyses are rerun and updated based on random data and initialization [I suggest you try this nifty feature].

The outline of this tutorial is as follows. Section 1 introduces the functions, and associated regression models, implemented by the `tgp` package, including plotting and visualization methods. The Bayesian mathematical specification of these models is contained in Section 2. In Section 3, the functions and methods implemented in the package are illustrated by example. The appendix covers miscellaneous topics such as how to link with the `ATLAS` libraries for fast linear algebra routines, and some of the details of implementation.

This document is intended as a tutorial, or initial guide, to the `tgp` package, covering key points, concepts, and methods. It was not meant to serve as an instruction manual. For more detailed documentation of the functions

R function	Ingredients	Description
blm	LLM	Linear Model
btlm	T	Linear CART
bgp	GP	GP Regression
bgpllm	GP, LLM	GP with jumps to the LLM
btgp	T, GP	treed GP Regression
btgpllm	T, GP, LLM	treed GP with jumps to the LLM
tgp		Master interface for the above methods

Table 1: Bayesian regression models implemented by the `tgp` package

contained in the package, see the package help-manuals. At an `R` prompt, type `help(package=tgp)`.

1 What is implemented?

The `tgp` package contains implementations of six Bayesian multivariate regression models, and functions for visualizing posterior predictive surfaces. These models, and the functions which implement them, are outlined in Section 1.1. Details pertaining to the mathematics of model specification, including prior and posterior distributions, is deferred to Section 2. Also implemented in the package are functions which aid in the sequential design of experiments for `tgp`-class models, which is what I call *adaptive sampling*. These functions are introduced at the end of this section.

1.1 Bayesian regression models

The six regression models implemented in this package are summarized in Table 1. They include combinations of treed partition models, (limiting) linear models, and Gaussian process models as indicated by T, LLM, & GP in the center—mixture of ingredients—column of the table. The model specification of each of these ingredients is contained in Section 2. Each is a fully Bayesian regression model, and in the table they are ordered by some notion of "flexibility". These `b*` functions, as I call them, are wrappers around the master `tgp` function which is an interface to `C` code implementing Bayesian treed Gaussian process models, with jumps to the limiting linear model (LLM). Each `b*` function implements a special case of the treed GP (`tgp`) model.

It is possible to invoke any of the `b*` methods directly via first calling the the `tgp.default.params` function and then the `tgp` function after some minor adjustments to the default parameterization. The help file for `tgp` shows how to do this for many of the examples in this document. The `b*` functions are intended as the sole interface to the Bayesian regression, so no further attention to the `tgp` master function will be included here. That is, with the exception of one example in Section 3.4 where a more custom model is needed in order to capture input dependent noise, and a remark that the easiest way to see how the

master tgp function implements one of the b* functions is to simply type the name of the function of interest into R. For example, to see the implementation of bgp, type:

```
> bgp
```

```
function (X, Z, XX = NULL, bprior = "bflat", corr = "expsep",
    BTE = c(1000, 4000, 3), R = 1, m0r1 = FALSE, pred.n = TRUE,
    ds2x = FALSE, ego = FALSE)
{
    n <- dim(X)[1]
    if (is.null(n)) {
        n <- length(X)
        X <- matrix(X, nrow = n)
        d <- 1
    }
    else {
        d <- dim(X)[2]
    }
    params <- tgp.default.params(d + 1)
    params$bprior <- bprior
    params$corr <- corr
    params$tree <- c(0, 0, 10)
    params$gamma <- c(0, 0.2, 0.7)
    return(tgp(X, Z, XX, BTE, R, m0r1, FALSE, params, pred.n,
        ds2x, ego))
}
```

The output (return-value) of tgp and the b* functions is a list-object of class "tgp". This is what is meant by a "tgp-class" object. This object retains all of the relevant information necessary to summarize posterior predictive inference, maximum *a' posteriori* (MAP) trees, and statistics for adaptive sampling. Information about its actual contents is contained in the help files for the tgp and b* functions. Generic print and plot methods are defined for tgp-class objects. The plot function is discussed below. The print function simply provides a list of the names of the fields comprising a tgp-class object.

This is as good a place as any to remark on the computational burdens of some of the modeling functions in this package. Fully Bayesian analyses with MCMC are not the "super-speediest" of all statistical models. Neither is inference for GP models, classical or Bayesian. Great care has been taken to make implementation of Bayesian inference of GP models as efficient as possible [See Appendix B]. However, inference for non-treed GPs for non-linear data is likely to be slow for data of more than a few hundred inputs. Two things are implemented by the package which can help speed things up a bit. The first is support for ATLAS for fast linear algebra. Details on linking this package with ATLAS is contained in Appendix A. The second is an argument called linburn to the tree class (T) functions in Table 1. When linburn = TRUE, the Markov

chain is initialized with a run of the Bayesian linear CART algorithm [4] prior to burn-in in order to pre-partition up the input space with linear models.

1.1.1 Plotting and visualization

The two main functions provided by the `tgp` package for visualization are `plot.tgp`, inheriting from the generic `plot` method, and a function called `tgp.trees` for graphical visualization of MAP trees as a function of the tree heights encountered while sampling from the Markov chain. I consider these functions to be visualization-"helper" functions. They provide useful, but very custom, visualizations.

The `plot.tgp` function can make plots in 1-d or 2-d. Of course, if the data are 1-d, the plot is in 1-d. If the data are 2-d, or higher, they are 2-d surface or perspective plots. Data which is 3-d, or higher, requires projection down to 2-d, or specification of a 2-d slice. The `plot.tgp` default is to make a projection onto the first two input variables. Alternate projections are specified as an argument (`proj`) to the function. Likewise, there is also an argument (`slice`) which allows one to specify which slice of the posterior predictive data is desired. For functions implementing models that use treed partitioning (those with a T in the center column of Table 1), the `plot.tgp` function will overlay the region boundaries of the MAP tree ($\hat{\mathcal{T}}$) found during MCMC.

A few of notes on 2-d plotting of `tgp` predictive output:

- 2-d plotting requires the `akima` package, available from CRAN. There is, in my opinion, a bug in the `akima` package, which produces `NA`'s when plotting data from a grid. For beautiful 2-d plots I suggest exporting the `tgp` predictive output to a text file and using `gnuplot`'s 2-d plotting features. See Chapter 4 of my thesis for examples [10]. Note that `gnuplot` expects gridded 2-d inputs to be encoded in a special "grid" format.

- Unfortunately, the current version of this package contains no examples—nor does this document—which demonstrate plotting of data with dimension larger than two. The example provided in Section 3.5 uses 10-d data, however no plottin is required. More examples will be included in future versions.

- The `plot.tgp` function, though limited many respects, has many more options than are illustrated [in Section 3] of this document. Please refer to the help files for more details.

The `tgp.trees` function provides a graphical representation of the MAP trees of each height encountered by the Markov chain during sampling. The function will not plot trees of height one, i.e., trees with no branching or partitioning. Plotting of trees requires the `maptree` package, which in turn requires the `combinat` package, both available from CRAN.

1.2 Sequential design of experiments

Sequential design of experiments, a.k.a. *adaptive sampling*, is not implemented by any *single* function in the `tgp` package. However, functions, and arguments to functions (and outputs from functions), have been included in order to facilitate the automation of adaptive sampling with `tgp`-class models. A detailed example is included in Section 3.6.

Arguments to `b*` functions, and `tgp`, which aid in adaptive sampling are `Ds2x` and `ego`. Both are booleans, i.e., should be set to `TRUE` or `FALSE` (the default is `FALSE`). `TRUE` booleans cause the `tgp`-class output list to contain vectors of the same name which contain statistics that can be used toward adaptive sampling. When `Ds2x = TRUE` then the $\Delta\sigma^2(\tilde{\mathbf{x}})$ statistic is computed at each $\tilde{\mathbf{x}} \in$ `XX`, in accordance the ALC (Active Learning–Cohn) algorithm [5]. Likewise, when `ego = TRUE`, statistics for Expected Global Optimization (EGO) are computed in order to asses the expected information gain for each $\tilde{\mathbf{x}} \in$ `XX` about the global minimum. The ALM (Active Learning–Mackay) algorithm is implemented by default in terms of difference in predictive quantiles for the inputs `XX`, which can be accessed via the `ZZ.q` output field. Details and references on the ALM, ALC, and EGO algorithms are provided in Section 2.

Calculation of EGO statistics is considered to be "alpha" functionality in this version of the `tgp` package. It has not been adequately tested, and its implementation is likely to change substationally in future versions of the package. There are also currently no examples which illustrate EGO adaptive sampling in Section 3.

The functions included in the package which explicitly aid in the sequential design of experiments are `tgp.design` and `dopt.gp`. They are both intended to produce sequential D-optimal candidate designs `XX` at which one or more of the adaptive sampling methods (ALM, ALC, EGO) can gather statistics. The `dopt.gp` function generates D-optimal candidates for a stationary Gaussian process. The `tgp.design` function extracts the MAP tree from a `tgp`-class object and uses `dopt.gp` on each region of the MAP partition in order to get treed sequential D-optimal candidates.

2 Methods and Models

This section provides a quick overview of the statistical models and methods implemented by the `tgp` package. Stationary Gaussian processes (GPs), GPs with jumps to the limiting linear model (LLM; a.k.a. GP LLM), treed partitioning for nonstationary models, and sequential design of experiments (a.k.a. *adaptive sampling*) concepts for these models are all briefly discussed. Appropriate references are provided for the details. Of course, the best reference is probably my thesis [10].

As a first pass on this document, it might make sense to skip this section and go straight on to the examples in Section 3.

2.1 Stationary Gaussian processes

Below is a hierarchical generative model for a stationary GP with linear tend for data $D = \{\mathbf{X}, \mathbf{Z}\}$.

$$\begin{aligned} \mathbf{Z}|\boldsymbol{\beta}, \sigma^2, \mathbf{K} &\sim N_n(\mathbf{F}\boldsymbol{\beta}, \sigma^2\mathbf{K}), \\ \boldsymbol{\beta}|\sigma^2, \tau^2, \mathbf{W}, \boldsymbol{\beta}_0 &\sim N_{m_X}(\boldsymbol{\beta}_0, \sigma^2\tau^2\mathbf{W}) \\ \boldsymbol{\beta}_0 &\sim N_{m_X}(\boldsymbol{\mu}, \mathbf{B}), \\ \sigma^2 &\sim IG(\alpha_\sigma/2, q_\sigma/2), \\ \tau^2 &\sim IG(\alpha_\tau/2, q_\tau/2), \\ \mathbf{W}^{-1} &\sim W((\rho\mathbf{V})^{-1}, \rho), \end{aligned} \tag{1}$$

where $\mathbf{F} = (\mathbf{1}, \mathbf{X})$, and $\mathbf{W}$ is a $(m_X + 1) \times (m_X + 1)$ matrix. N, IG, and W are the (Multivariate) Normal, Inverse-Gamma, and Wishart distributions, respectively. Constants $\boldsymbol{\mu}, \mathbf{B}, \mathbf{V}, \rho, \alpha_\sigma, q_\sigma, \alpha_\tau, q_\tau.$ are treated as known.

The GP correlation structure $\mathbf{K}$ is chosen either from the isotropic power family, or separable power family, with a fixed power p_0 (see below), but unknown (random) range and nugget parameters. Correlation functions used in the `tgp` package take the form $K(\mathbf{x}_j, \mathbf{x}_k) = K^*(\mathbf{x}_j, \mathbf{x}_k) + g\delta_{j,k}$, where $\delta_{\cdot,\cdot}$ is the Kronecker delta function, and K^* is a *true* correlation representative from a parametric family.

All parameters in (1) can be sampled using Gibbs steps, except for the covariance structure and nugget parameters, and their hyperparameters, which can be sampled via Metropolis-Hastings [11, 10].

2.1.1 The nugget

The g term in the correlation function $K(\cdot, \cdot)$ is referred to as the *nugget* in the geostatistics literature [15, 6] and sometimes as *jitter* in the Machine Learning literature [16]. It must always be positive ($g > 0$), and serves two purposes. Primarily, it provides a mechanism for introducing measurement error into the stochastic process. It arises when considering a model of the form:

$$Z(\mathbf{X}) = m(\mathbf{X}, \boldsymbol{\beta}) + \varepsilon(\mathbf{X}) + \eta(\mathbf{X}), \tag{2}$$

where $m(\cdot, \cdot)$ is underlying (usually linear) mean process, $\varepsilon(\cdot)$ is a process covariance whose underlying correlation is governed by K^*, and $\eta(\cdot)$ is simply Gaussian noise. Secondarily, though perhaps of equal practical importance, the nugget (or jitter) prevents $\mathbf{K}$ from becoming numerically singular. Notational convenience and conceptual congruence motivates referral to $\mathbf{K}$ as a correlation matrix, even though the nugget term (g) forces $K(\mathbf{x}_i, \mathbf{x}_i) > 1$.

2.1.2 Exponential Power family

Correlation functions in the *isotropic power* family are *stationary* which means that correlations are measured identically throughout the input domain, and

isotropic in that correlations $K^*(\mathbf{x}_j, \mathbf{x}_k)$ depend only on a function of the Euclidean distance between $\mathbf{x}_j$ and $\mathbf{x}_k$: $||\mathbf{x}_j - \mathbf{x}_k||$.

$$K^*_\nu(\mathbf{x}_j, \mathbf{x}_k | d_\nu) = \exp\left\{ -\frac{||\mathbf{x}_j - \mathbf{x}_k||^{p_0}}{d} \right\}, \tag{3}$$

where $d > 0$ is referred to as the *width* or *range* parameter. The power $0 < p_0 \leq 2$ determines the smoothness of the underlying process. A typical default choice is the Gaussian $p_0 = 2$ which gives an infinitely differentiable process.

A straightforward enhancement to the isotropic power family is to employ a unique range parameter d_i in each dimension ($i = 1, \ldots, m_X$). The resulting correlation function is still stationary, but no longer isotropic.

$$K^*(\mathbf{x}_j, \mathbf{x}_k | \mathbf{d}) = \exp\left\{ -\sum_{i=1}^{m_X} \frac{|x_{ij} - x_{ik}|^{p_0}}{d_i} \right\} \tag{4}$$

The (non-separable) isotropic power family is a special case (when $d_i = d$, for $i = 1, \ldots, m_X$). With the separable power family, one can model correlations in some input variables as stronger than others. However, with added flexibility comes added costs. When the true underlying correlation structure is isotropic, estimating the extra parameters of the separable model represents a sort of overkill.

2.1.3 Prediction and Adaptive Sampling

The predicted value of $z(\mathbf{x})$ is normally distributed with mean and variance

$$\hat{z}(\mathbf{x}) = \mathbf{f}^\top(\mathbf{x})\tilde{\boldsymbol{\beta}} + \mathbf{k}(\mathbf{x})^\top \mathbf{K}^{-1}(\mathbf{Z} - \mathbf{F}\tilde{\boldsymbol{\beta}}), \tag{5}$$

$$\hat{\sigma}^2(\mathbf{x}) = \sigma^2[\kappa(\mathbf{x}, \mathbf{x}) - \mathbf{q}^\top(\mathbf{x})\mathbf{C}^{-1}\mathbf{q}(\mathbf{x})], \tag{6}$$

where $\tilde{\boldsymbol{\beta}}$ is the posterior mean estimate of $\boldsymbol{\beta}$, and

$$\begin{aligned} \mathbf{C}^{-1} &= (\mathbf{K} + \mathbf{F}\mathbf{W}\mathbf{F}^\top/\tau^2)^{-1} \\ \mathbf{q}(\mathbf{x}) &= \mathbf{k}(\mathbf{x}) + \tau^2 \mathbf{F}\mathbf{W}\mathbf{f}(\mathbf{x}) \\ \kappa(\mathbf{x}, \mathbf{y}) &= K(\mathbf{x}, \mathbf{y}) + \tau^2 \mathbf{f}^\top(\mathbf{x})\mathbf{W}\mathbf{f}(\mathbf{y}) \end{aligned}$$

with $\mathbf{f}^\top(\mathbf{x}) = (1, \mathbf{x}^\top)$, and $\mathbf{k}(\mathbf{x})$ a $n-$vector with $\mathbf{k}_{\nu,j}(\mathbf{x}) = K(\mathbf{x}, \mathbf{x}_j)$, for all $\mathbf{x}_j \in \mathbf{X}$. Notice that $\hat{\sigma}(\mathbf{x})^2$ does not directly depend on the observed responses $\mathbf{Z}$. These equations often called *kriging* equations [15].

The ALM algorithm is implemented with MCMC inference by computing the norm (or width) of predictive quantiles obtained by samples from the Normal distribution given above. The ALC algorithm computes the reduction in variance given that the candidate location $\tilde{\mathbf{x}} \in \tilde{\mathbf{X}}$ is added into the data (averaged

over a reference set $\tilde{\mathbf{Y}}$):

$$\Delta\hat{\sigma}^2(\tilde{\mathbf{x}}) = \frac{1}{|\tilde{\mathbf{Y}}|}\sum_{\mathbf{y}\in\tilde{\mathbf{Y}}} \Delta\hat{\sigma}^2_{\mathbf{y}}(\tilde{\mathbf{x}}) = \frac{1}{|\tilde{\mathbf{Y}}|}\sum_{\mathbf{y}\in\tilde{\mathbf{Y}}} \hat{\sigma}^2_{\mathbf{y}} - \hat{\sigma}^2_{\mathbf{y}}(\tilde{\mathbf{x}}) \tag{7}$$
$$= \frac{1}{|\tilde{\mathbf{Y}}|}\sum_{\mathbf{y}\in\tilde{\mathbf{Y}}} \frac{\sigma^2\left[\mathbf{q}_N^\top(\mathbf{y})\mathbf{C}_N^{-1}\mathbf{q}_N(\tilde{\mathbf{x}}) - \kappa(\tilde{\mathbf{x}},\mathbf{y})\right]^2}{\kappa(\tilde{\mathbf{x}},\tilde{\mathbf{x}}) - \mathbf{q}_N^\top(\tilde{\mathbf{x}})\mathbf{C}_N^{-1}\mathbf{q}_N(\tilde{\mathbf{x}})},$$

which is easily computed using MCMC methods. In the `tgp` package, the reference set is taken to be the same as the candidate set, i.e., $\tilde{\mathbf{Y}} = \tilde{\mathbf{X}}$.

The Expected Global Optimization (EGO) algorithm is centered around a statistic which captures the expected improvement in the model about its ability to predict the spatial location of its global minimum. If $f_{\min}$ is the model's current best guess about the minimum, e.g., $f_{\min} = \min\{z_1, \ldots, z_N\}$, then the expected improvement at the point $\tilde{\mathbf{x}}$ can reasonably be encoded as

$$E[I(\tilde{\mathbf{x}})] = E[\max(f_{\min} - Z(\tilde{\mathbf{x}}), 0)],$$

which, after a tedious integration by parts, can be shown to work out to be

$$E[I(\tilde{\mathbf{x}})] = (f_{\min} - \hat{z}(\tilde{\mathbf{x}}))\Phi\left(\frac{f_{\min} - \hat{z}(\tilde{\mathbf{x}})}{\hat{\sigma}^2(\tilde{\mathbf{x}})}\right) + \hat{\sigma}^2(\tilde{\mathbf{x}})\phi\left(\frac{f_{\min} - \hat{z}(\tilde{\mathbf{x}})}{\hat{\sigma}^2(\tilde{\mathbf{x}})}\right) \tag{8}$$

where $\hat{z}$ and $\hat{\sigma}^2$ are taken from the equations for the posterior predictive distribution (5). Φ and ϕ are the standard Normal cumulative distribution and probability density functions, respectively. MCMC samples from (8) can be gathered in order to determine which $\tilde{\mathbf{x}}$ of a candidate set of locations $\tilde{\mathbf{x}} \in \tilde{\mathbf{X}}$ give the highest reduction in uncertainty about the global minimum.

2.2 GPs and Limiting linear models

A special limiting case of the Gaussian process model is the standard linear model. Replacing the top (likelihood) line in the hierarchical model (1)

$$\mathbf{Z}|\boldsymbol{\beta}, \sigma^2, \mathbf{K} \sim N(\mathbf{F}\boldsymbol{\beta}, \sigma^2\mathbf{K}) \qquad \text{with} \qquad \mathbf{Z}|\boldsymbol{\beta}, \sigma^2 \sim N(\mathbf{F}\boldsymbol{\beta}, \sigma^2\mathbf{I}),$$

where $\mathbf{I}$ is the $n \times n$ identity matrix, gives a parameterization of a linear model. From a phenomenological perspective, GP regression is more flexible than standard linear regression in that it can capture nonlinearities in the interaction between covariates ($\mathbf{x}$) and responses (z). From a modeling perspective, the GP can be more than just overkill for linear data. Parsimony and over-fitting considerations are just the tip of the iceberg. It is also unnecessarily computationally expensive, as well as numerically unstable. Specifically, it requires the inversion of a large covariance matrix— an operation whose computing cost grows with the cube of the sample size. Moreover, large finite d parameters can be problematic from a numerical perspective because, unless g is also large, the resulting covariance matrix can be numerically singular when the off-diagonal elements of $\mathbf{K}$ are nearly one.

Bayesians can take advantage of the limiting linear model (LLM) by constructing prior for the "mixture" of the GP with its LLM [12, 10]. The key idea is an augmentation of the parameter space by m_X indicators $\mathbf{b} = \{b\}_{i=1}^{m_X} \in \{0,1\}^{m_X}$. The boolean b_i is intended to select either the GP ($b_i = 1$) or its LLM for the i^{th} dimension. The actual range parameter used by the correlation function is multiplied by $\mathbf{b}$: e.g. $K^*(\cdot,\cdot|\mathbf{b}^\top\mathbf{d})$. To encode the preference that GPs with larger range parameters be more likely to "jump" to the LLM, the prior on b_i is specified as a function of the range parameter d_i: $p(b_i, d_i) = p(b_i|d_i)p(d_i)$.

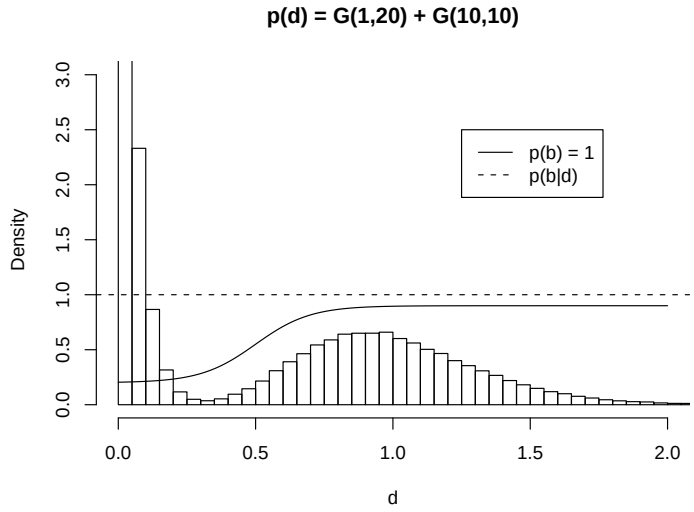


Figure 1: Prior distribution for the boolean (b) superimposed on $p(d)$.

Probability mass functions which increase as a function of d_i, e.g.,

$$p_{\gamma,\theta_1,\theta_2}(b_i = 0|d_i) = \theta_1 + (\theta_2 - \theta_1)/(1 + \exp\{-\gamma(d_i - 0.5)\}) \tag{9}$$

with $0 < \gamma$ and $0 \le \theta_1 \le \theta_2 < 1$, can encode such a preference by calling for the exclusion of dimensions i with with large d_i when constructing $\mathbf{K}$. Thus b_i determines whether the GP or the LLM is in charge of the marginal process in the i^{th} dimension. Accordingly, θ_1 and θ_2 represent minimum and maximum probabilities of jumping to the LLM, while γ governs the rate at which $p(b_i = 0|d_i)$ grows to θ_2 as d_i increases. Figure 1 plots $p(b_i = 0|d_i)$ for $(\gamma, \theta_1, \theta_2) = (10, 0.2, 0.95)$ superimposed on a convenient $p(d_i)$ which is taken to be a mixture of Gamma distributions,

$$p(d) = [G(d|\alpha = 1, \beta = 20) + G(d|\alpha = 10, \beta = 10)]/2, \tag{10}$$

representing a population of GP parameterizations for wavy surfaces (small d) and a separate population of those which are quite smooth or approximately

linear. The θ_2 parameter is taken to be strictly less than one so as not to preclude a GP which models a genuinely nonlinear surface using an uncommonly large range setting.

The implied prior probability of the full m_X-dimensional LLM is

$$p(\text{linear model}) = \prod_{i=1}^{m_X} p(b_i = 0|d_i) = \prod_{i=1}^{m_X} \left[\theta_1 + \frac{\theta_2 - \theta_1}{1 + \exp\{-\gamma(d_i - 0.5)\}}\right]. \quad (11)$$

Notice that the resulting process is still a GP if any of the booleans b_i are one. The primary computational advantage associated with the LLM is foregone unless all of the b_i's are zero. However, the intermediate result offers increased numerical stability and represents a unique transitionary model lying somewhere between the GP and the LLM. It allows for the implementation of semiparametric stochastic processes like $Z(\mathbf{x}) = \boldsymbol{\beta} f(\mathbf{x}) + \varepsilon(\tilde{\mathbf{x}})$ representing a piecemeal spatial extension of a simple linear model. The first part ($\boldsymbol{\beta} f(\mathbf{x})$) of the process is linear in some known function of the full set of covariates $\mathbf{x} = \{x_i\}_{i=1}^{m_X}$, and $\varepsilon(\cdot)$ is a spatial random process (e.g. a GP) which acts on a subset of the covariates $\tilde{\mathbf{x}}$. Such models are commonplace in the statistics community [7]. Traditionally, $\tilde{\mathbf{x}}$ is determined and fixed *a' priori*. The separable boolean prior (9) implements an adaptively semiparametric process where the subset $\tilde{\mathbf{x}} = \{x_i : b_i = 1, i = 1, \ldots, m_X\}$ is given a prior distribution, instead of being fixed.

2.2.1 Prediction and Adaptive Sampling under LLM

Prediction under the limiting GP model is a simplification of (5) when it is known that $\mathbf{K} = (1 + g)\mathbf{I}$. It can be shown [12, 10] that the predicted value of z at $\mathbf{x}$ is normally distributed with mean $\hat{z}(\mathbf{x}) = \mathbf{f}^\top(\mathbf{x})\tilde{\boldsymbol{\beta}}$ and variance $\hat{\sigma}(\mathbf{x})^2 = \sigma^2[1 + \mathbf{f}^\top(\mathbf{x})\mathbf{V}_{\tilde{\beta}}\mathbf{f}(\mathbf{x})]$, where $\mathbf{V}_{\tilde{\beta}} = (\tau^{-2} + \mathbf{F}^\top\mathbf{F}(1+g))^{-1}$. This is preferred over (5) with $\mathbf{K} = \mathbf{I}(1 + g)$ because an $m_X \times m_X$ inversion is preferred over an $n \times n$ one.

Applying the ALC algorithm under the LLM is computationally less intense compared to ALC under a full GP. Starting with the predictive variance given in (5), the expected reduction in variance under the LM is [10]

$$\Delta\hat{\sigma}^2_{\mathbf{y}}(\mathbf{x}) = \frac{\sigma^2[\mathbf{f}^\top(\mathbf{y})\mathbf{V}_{\tilde{\beta}_N}\mathbf{f}(\mathbf{x})]^2}{1 + g + \mathbf{f}^\top(\mathbf{x})\mathbf{V}_{\tilde{\beta}_N}\mathbf{f}(\mathbf{x})}. \quad (12)$$

Since only an $m_X \times m_X$ inverse is required, Eq. (12) is preferred over simply replacing $\mathbf{K}$ with $\mathbf{I}(1 + g)$ in (7), which requires an $n \times n$ inverse.

The statistic for the EGO algorithm is the same under the LLM as (8) for the GP. Of course, it helps to use the linear predictive equations instead of the kriging ones for $\hat{z}(\mathbf{x})$ and $\hat{\sigma}^2(\mathbf{x})$.

2.3 Treed partitioning

Nonstationary models are obtained by treed partitioning and inferring a separate model within each region of the partition. Treed partitioning is accomplished

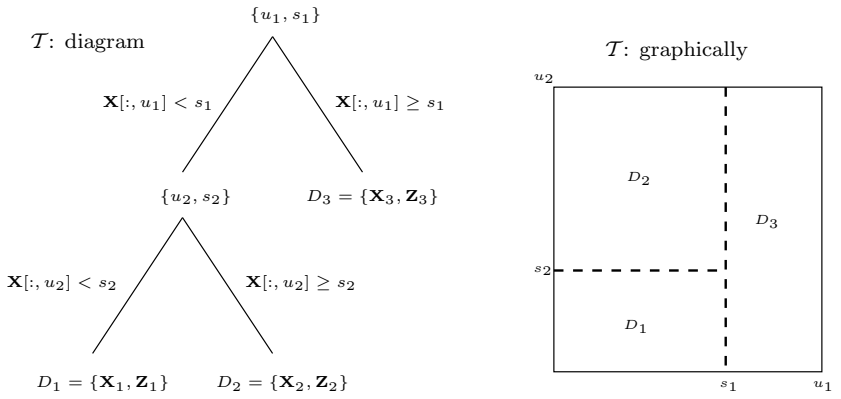


Figure 2: An example tree $\mathcal{T}$ with two splits, resulting in three regions, shown in a diagram (*left*) and pictorially (*left*).

by making (recursive) binary splits on the value of a single variable so that region boundaries are parallel to coordinate axes. Partitioning is recursive, so each new partition is a sub-partition of a previous one. Since variables may be revisited, there is no loss of generality by using binary splits as multiple splits on the same variable are equivalent to a non-binary split.

Figure 2 shows an example tree. In this example, region D_1 contains $\mathbf{x}$'s whose u_1 coordinate is less than s_1 and whose u_2 coordinate is less than s_2. Like D_1, D_2 has $\mathbf{x}$'s whose coordinate u_1 is less than s_1, but differs from D_1 in that the u_2 coordinate must be bigger than or equal to s_2. Finally, D_3 contains the rest of the $\mathbf{x}$'s differing from those in D_1 and D_2 because the u_1 coordinate of its $\mathbf{x}$'s is greater than or equal to s_1. The corresponding response values (z) accompany the $\mathbf{x}$'s of each region.

These sorts of models are often referred to as Classification and Regression Trees (CART) [1]. CART has become popular because of its ease of use, clear interpretation, and ability to provide a good fit in many cases. The Bayesian approach is straightforward to apply to tree models, provided that one can specify a meaningful prior for the size of the tree. The tree process implemented in the `tgp` package follows Chipman et al. [3] who specify the prior through a tree-generating process. Starting with a null tree (all data in a single partition), the tree, $\mathcal{T}$, is probabilistically split recursively with each partition, η, being split with probability $p_{\text{SPLIT}}(\eta, \mathcal{T}) = a(1 + q_\eta)^{-b}$ where q_η is the depth of η in $\mathcal{T}$ and a and b are parameters chosen to give an appropriate size and spread to the distribution of trees.

Extending the work of Chipman et al. [4], the `tgp` package implements a stationary GP with linear trend, or GP LLM, independently within each of the regions depicted by a tree $\mathcal{T}$. Integrating out dependence on $\mathcal{T}$ is accomplished by reversible-jump MCMC (RJ-MCMC) via tree operations *grow, prune, change*, and *swap* [3]. To keep things simple, proposals for new parameters—via

an increase in the number of partitions—are drawn from their priors, thus eliminating the Jacobian term usually present in RJ-MCMC. New splits are chosen uniformly from the set of marginalized input locations $\mathbf{X}$. The *swap* operation has been augmented with a *rotate* option to improve mixing of the Markov chain [10].

There are many advantages to partitioning the input space into regions, and fitting separate GPs (or GP LLMs) within each region. Partitioning allows for the modeling of non-stationary behavior, and can ameliorate some of the computational demands by fitting models to less data. Finally, a fully Bayesian approach yields a uniquely efficient nonstationary, nonparametric, or semiparametric (in the case of the GP LLM) regression tool.

2.4 (Treed) sequential D-optimal design

In the statistics community, the traditional approach to sequential data solicitation goes under the general heading of *(Sequential) Design of Experiments* [19]. Depending on a choice of utility, different algorithms for obtaining optimal designs can be derived. For example, one can choose the Kullback-Leibler distance between the posterior and prior distributions as a utility. For Gaussian process models with correlation matrix $\mathbf{K}$, this is equivalent to maximizing $\det(\mathbf{K})$. Subsequently chosen input configurations are called $D-$optimal designs. Choosing quadratic loss leads to what are called $A-$optimal designs. An excellent review of Bayesian approaches to the design of experiments is provided by Chaloner & Verdinelli [2].

Other approaches used by the statistics community include space-filling designs: e.g. max-min distance and Latin Hypercube (LH) designs [19]. The `FIELDS` package [8], available from CRAN, implements code for space-filling designs in addition to kriging and thin plate spline models for spatial interpolation.

A hybrid approach to designing experiments employs active learning techniques. The idea is to choose a set of candidate input configurations $\tilde{\mathbf{X}}$ (say, a $D-$optimal or LH design) and an active learning rule for determining the order in which they are be added into the design. The ALM algorithm has been shown to approximate maximum expected information designs by selecting the candidate location $\tilde{\mathbf{x}} \in \tilde{\mathbf{X}}$ which has the greatest standard deviation in predicted output [14]. An alternative algorithm is to select $\tilde{\mathbf{x}}$ minimizing the resulting expected squared error averaged over the input space [5], called ALC for Active Learning–Cohn. Seo et al. [20] provide a comparison between ALC and ALM using standard GPs. The EGO algorithm can be used to find global minima.

Choosing candidate configurations $\tilde{\mathbf{X}}$ (`XX` in the `tgp` package), at which to gather ALM, ALC, or EGO statistics, is half of the challenge in the hybrid approach to experimental design. Arranging candidates so that they are well-spaced out relative to themselves, and relative to already sampled configurations, is clearly desirable. Towards this end, a sequential D-optimal design is a good first choice. However, traditional D-optimal designs fall short of the task for a number of reasons. They are based on a *known* parameterization of a single GP model, and are thus not well-suited to MCMC inference. A D-

optimal design may not choose candidates in the "interesting" part of the input space, because sampling is high there already. Classic optimal design criteria, in general, are ill-suited partition models, wherein "closeness" may not measured homogeneously across the input space. Another disadvantage is computational, namely decomposing and finding the determinant of a large covariance matrix.

One possible solution to both computational and nonstationary modeling issues is to use treed sequential D-optimal design. Separate sequential D-optimal designs can be computed in each of the partitions depicted by the maximum *a posteriori* (MAP) tree $\hat{\mathcal{T}}$. The number of candidates selected from each region can be proportional to the volume of—or proportional to the number of grid locations in—the region. MAP parameters $\hat{\boldsymbol{\theta}}_\nu|\hat{\mathcal{T}}$, or "neutral" or "exploration encouraging" ones, can be used to create the candidate design. Separating design from inference by using custom parameterizations in design steps, rather than inferred ones, is a common practice [19]. Small range parameters, for learning about the wiggliness of the response, and a modest nugget parameter, for numerical stability, tend to work well together.

Finding a local maxima is generally sufficient to get well-spaced candidates. The `dopt.gp` function uses a stochastic ascent algorithm which can find local maxima without calculating too many determinants

3 Examples using `tgp`

The following subsections take the reader through a series of examples based, mostly, on synthetic data. At least two different `b*` models are fit for each set of data, offering comparisons, and contrasts. Duplicating these examples in your own `R` session is highly recommended. The `Stangle` function can help extract executable `R` code from this document. For example, extract the code for the exponential data of Section 3.3 with one command.

```
> Stangle(vignette("exp", package="tgp")$file))
```

This will write a file called "exp.R". Additionally, each of the subsections that follow is available as an `R` demo. Try `demo(package="tgp")` for a listing of available demos. To envoke the demo for the exponential data of Section 3.3 try `demo(exp, package="tgp")`. This is equivalent to `source("exp.R")` because the demos were created using `Stangle` on the source files of this document.

Other successful uses of the methods in this pacakge include applications to the Boston housing data [13], and designing an experiment for a reusable NASA launch vehicle [11, 10] called the Langely glide-back booster (LGBB). These are not included as exmaples here. The Boston housing data is a large data set. Repeating the experiment of Chipman et al. [4] is computationally intensive, and impractical for this document [12, 10]. The LGBB experiment is also a big one, involves propriatary data, and utilizes ab sophicistated adaptive sampling interface to NASA supercomputers for the evaluations of computational fluid dynamics codes for the online accumulation of adaptively sampled input configurations [11, 10].

3.1 1-d Linear data

Consider data sampled from a linear model.

$$z_i = 1 + 2x_i + \epsilon, \quad \text{where} \quad \epsilon_i \overset{\text{iid}}{\sim} N(0, 0.25^2) \tag{13}$$

The following `R` code takes a sample $\{\mathbf{X}, \mathbf{Z}\}$ of size $N = 50$ from (13). It also chooses $N' = 99$ evenly spaced predictive locations $\tilde{\mathbf{X}} =$ `XX`.

```
> X <- seq(0, 1, length = 50)
> XX <- seq(0, 1, length = 99)
> Z <- 1 + 2 * X + rnorm(length(X), sd = 0.25)
```

Using `tgp` on this data with a Bayesian hierarchical linear model goes as follows:

```
> lin.blm <- blm(X = X, XX = XX, Z = Z)

state = 357 144 97 ignored, using R RNG
n=50, d=1, nn=99, BTE=(1000,4000,3), R=1, linburn=0
predicting at data locations
correlation: separable power exponential
linear prior: flat
starting d=0.5, nug=0.1, s2=1, tau2=1
starting beta = 0 0
tree[alpha,beta]=[0,0], minpart=10
s2[a0,g0]=[5,10]
d[a,b][0,1]=[1,20],[10,10]
nug[a,b][0,1]=[1,1],[1,1]
gamlin = [-1,0.2,0.7]
fixing d prior
fixing nug prior
s2 lambda[a0,g0]=[0.2,10]

burn in:
r=1000 corr=[0] : n = 50

Obtaining samples (nn=99 predictive locations):
r=1000 corr=[0] : mh=1 n = 50
r=2000 corr=[0] : mh=1 n = 50
r=3000 corr=[0] : mh=1 n = 50

finished repetition 1 0f 1
removed 0 leaves from the tree
```

The first group of text printed to `stdout` is a summary of the prior parameterization. Then, MCMC progress indicators are printed every 1,000 rounds. The linear model is indicated by `cor=[0]`.The GUI versions of `R`, on `Windows` or

```
> plot(lin.blm, main = "Linear Model,")
> abline(1, 2, lty = 3, col = "blue")
```

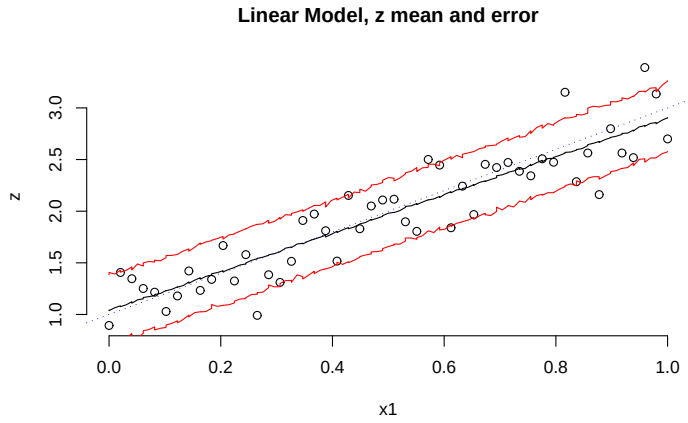


Figure 3: Posterior predictive distribution using `blm` on synthetic linear data: mean and 90% credible interval. The actual generating lines are shown as blue-dotted.

`MacOS X`, usually buffers `stdout`, rendering this feature essentially useless. In terminal versions, e.g. `Unix`, the progress indicators can give a sense of when the code will finish. Also note that a user cannot interact while the `C` code is running. This will be changed in future versions.

The generic `plot` method can be used to visualize the fitted posterior predictive surface in terms of means and credible intervals. Figure 3 shows how to do it, and what you get.

If, say, you were unsure about the dubious "linearness" of this data, you might try a GP LLM (using `btgpllm`) and let a more flexible model speak as to the linearity of the process.

```
> lin.gpllm <- bgpllm(X = X, XX = XX, Z = Z)

state = 474 363 977 ignored, using R RNG
n=50, d=1, nn=99, BTE=(2000,7000,2), R=1, linburn=0
predicting at data locations
correlation: separable power exponential
linear prior: flat
starting d=0.5, nug=0.1, s2=1, tau2=1
starting beta = 0 0
tree[alpha,beta]=[0,0], minpart=10
s2[a0,g0]=[5,10]
d[a,b][0,1]=[1,20],[10,10]
nug[a,b][0,1]=[1,1],[1,1]
gamlin = [10,0.2,0.7]
fixing d prior
```

```
fixing nug prior
s2 lambda[a0,g0]=[0.2,10]

burn in:
r=1000 corr=[0] : n = 50
r=2000 corr=[0] : n = 50

Obtaining samples (nn=99 predictive locations):
r=1000 corr=[0.788014] : mh=1 n = 50
r=2000 corr=[0] : mh=1 n = 50
r=3000 corr=[0] : mh=1 n = 50
r=4000 corr=[0.982654] : mh=1 n = 50
r=5000 corr=[0] : mh=1 n = 50

finished repetition 1 0f 1
removed 0 leaves from the tree
```

```
> plot(lin.gpllm, main = "GP LLM,")
> abline(1, 2, lty = 4, col = "blue")
```

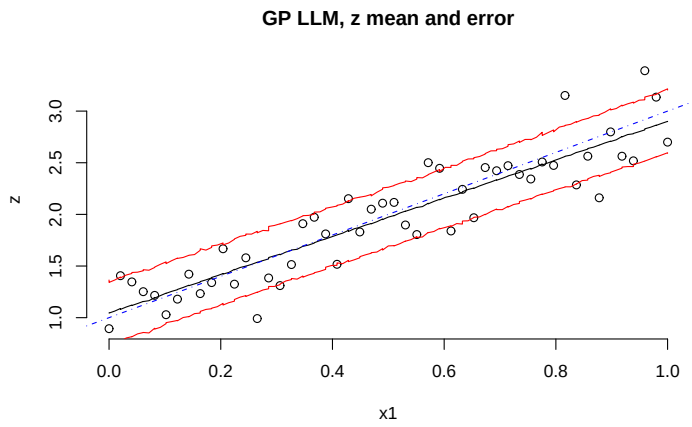


Figure 4: Posterior predictive distribution using `bgpllm` on synthetic linear data: mean and 90% credible interval. The actual generating lines are shown as blue-dotted.

Whenever the progress indicators show `corr[0]` the process is under the LLM in that round, and the GP otherwise. A plot of the resulting surface is shown in Figure 4 for comparison. Since the data is linear, the resulting predictive surfaces should look strikingly similar to one another. On occasion, the GP LLM may find some bendy-ness in the surface. This happens rarely with samples as large as $N = 50$, but is quite a bit more common for $N < 20$.

3.2 1-d Synthetic Sine Data

Consider 1-dimensional simulated data which is partly a mixture of sines and cosines, and partly linear.

$$z(x) = \begin{cases} \sin\left(\frac{\pi x}{5}\right) + \frac{1}{5}\cos\left(\frac{4\pi x}{5}\right) & x < 10 \\ x/10 - 1 & \text{otherwise} \end{cases} \tag{14}$$

The `R` code below obtains $N = 100$ evenly spaced samples from this data in the domain $[0, 20]$, with noise added to keep things interesting.

```
> X <- seq(0, 20, length = 100)
> XX <- seq(0, 20, length = 99)
> Z <- (sin(pi * X/5) + 0.2 * cos(4 * pi * X/5)) * (X <=
+     9.6)
> lin <- X > 9.6
> Z[lin] <- -1 + X[lin]/10
> Z <- Z + rnorm(length(Z), sd = 0.1)
```

Some evenly spaced predictive locations `XX` are also created. By design, the data is clearly nonstationary. Not knowing this, good first model choice for this data might be a GP, since it is clearly nonlinear.

```
> sin.bgp <- bgp(X = X, Z = Z, XX = XX)

> plot(sin.bgp, main = "GP,")
```

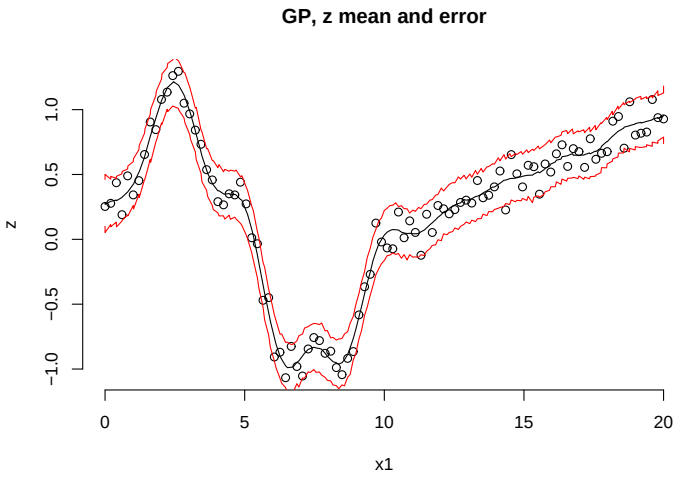


Figure 5: Posterior predictive distribution using `bgp` on synthetic sinusoidal data: mean and 90% credible interval

Progress indicators have been suppressed. Figure 5 shows the resulting posterior predictive surface under the GP. Notice how the (stationary) GP gets the wiggliness of the sinusoidal region, but fails to capture the smoothness of the linear region. This is becuase the data comes from a process that is nonstationary.

So one might consider a Bayesian CART model instead.

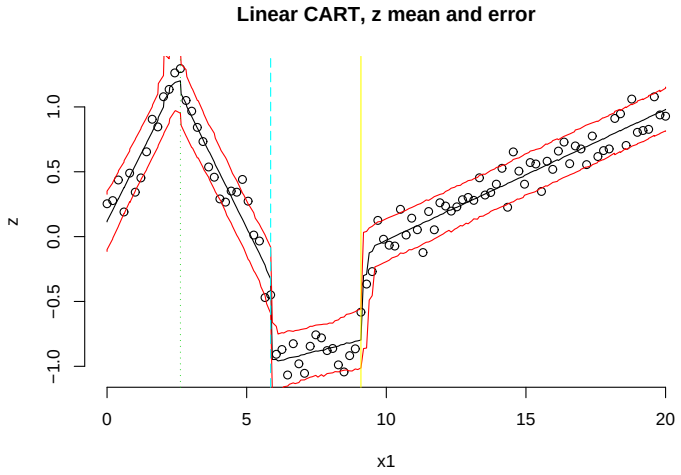
```
> sin.btlm <- btlm(X = X, Z = Z, XX = XX)

state = 888 23 373 ignored, using R RNG
n=100, d=1, nn=99, BTE=(2000,7000,2), R=1, linburn=0
predicting at data locations
correlation: separable power exponential
linear prior: flat
starting d=0.5, nug=0.1, s2=1, tau2=1
starting beta = 0 0
tree[alpha,beta]=[0.25,2], minpart=10
s2[a0,g0]=[5,10]
d[a,b][0,1]=[1,20],[10,10]
nug[a,b][0,1]=[1,1],[1,1]
gamlin = [-1,0.2,0.7]
fixing d prior
fixing nug prior
s2 lambda[a0,g0]=[0.2,10]

burn in:
**GROW(0,0)<-0** @depth 0: [0,0.474747], n=(48,52)
**GROW(0,0)<-0** @depth 1: [0,0.161616], n=(17,29)
**GROW(0,0)<-0** @depth 2: [0,0.272727], n=(11,18)
r=1000 corr=[0] [0] [0] [0] : n = 15 15 16 54
r=2000 corr=[0] [0] [0] [0] : n = 13 17 16 54

Obtaining samples (nn=99 predictive locations):
r=1000 corr=[0] [0] [0] [0] : mh=4 n = 10 20 16 54
r=2000 corr=[0] [0] [0] [0] : mh=4 n = 14 16 16 54
r=3000 corr=[0] [0] [0] [0] : mh=4 n = 14 16 16 54
r=4000 corr=[0] [0] [0] [0] : mh=4 n = 14 16 17 53
r=5000 corr=[0] [0] [0] [0] : mh=4 n = 13 17 16 54
Grow: 0.008174%, Prune: 0%, Change: 0.3092%, Swap: 0.7701%

finished repetition 1 0f 1
removed 4 leaves from the tree
```

MCMC progress indicators printed to `stdout` indicate successful *grow* and *prune* operations as they happen, and region sizes n every 1,000 rounds.

Figure 6 shows the resulting posterior predictive surface (*top*) and trees (*bottom*). The MAP partition ($\hat{\mathcal{T}}$) is also drawn onto the surface plot (*top*) in the form of vertical lines. The CART model captures the smoothness of the linear region just fine, but comes up short in the sinusoidal region—doing the best it can with piecewise linear models.

The ideal model for this data is the Bayesian treed GP becuase it can be both smooth and wiggly.

```
> plot(sin.btlm, main = "Linear CART,")
```




```
> tgp.trees(sin.btlm)
```

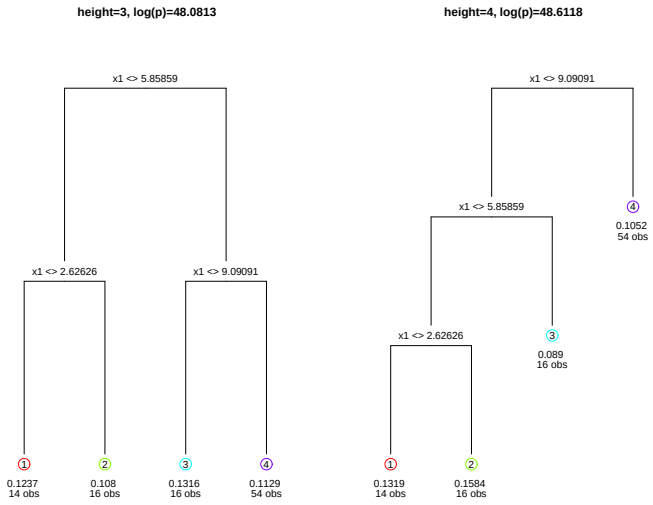
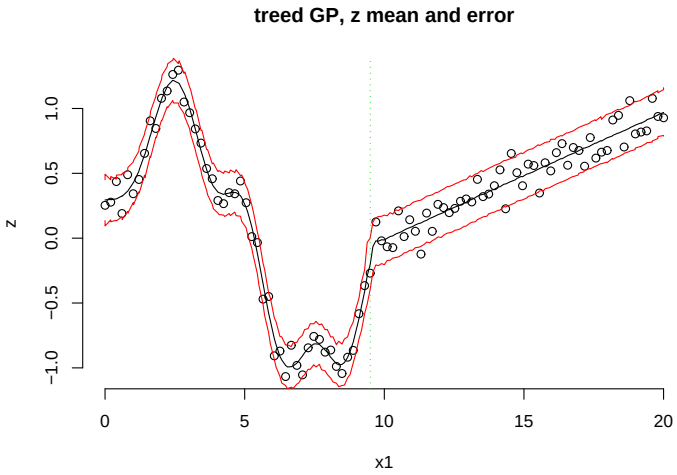


Figure 6: *Top:* Posterior predictive distribution using `btlm` on synthetic sinusoidal data: mean and 90% credible interval, and MAP partition ($\hat{T}$); *Bottom* MAP trees for each height encountered in the Markov chain.

```
> sin.btgp <- btgp(X = X, Z = Z, XX = XX)
```

Progress indicators have been suppressed. Figure 7 shows the resulting posterior predictive surface (*top*) and trees (*bottom*).




```
> tgp.trees(sin.btgp)
```

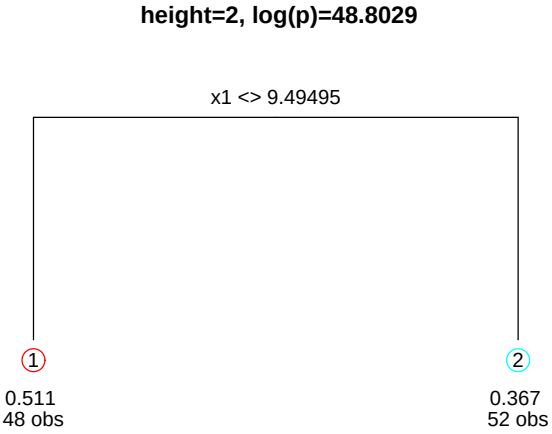


Figure 7: *Top:* Posterior predictive distribution using `btgp` on synthetic sinusoidal data: mean and 90% credible interval, and MAP partition ($\hat{T}$); *Bottom* MAP trees for each height encountered in the Markov chain.

Finally, speedups can be obtained if the GP is allowed to jump to the LLM [10], since half of the response surface is *very* smooth, or linear.

```
> sin.btgpllm <- btgpllm(X = X, Z = Z, XX = XX)

state = 477 854 556 ignored, using R RNG
n=100, d=1, nn=99, BTE=(2000,7000,2), R=1, linburn=0
```

```
predicting at data locations
correlation: separable power exponential
linear prior: flat
starting d=0.5, nug=0.1, s2=1, tau2=1
starting beta = 0 0
tree[alpha,beta]=[0.25,2], minpart=10
s2[a0,g0]=[5,10]
d[a,b][0,1]=[1,20],[10,10]
nug[a,b][0,1]=[1,1],[1,1]
gamlin = [10,0.2,0.7]
fixing d prior
fixing nug prior
s2 lambda[a0,g0]=[0.2,10]

burn in:
**GROW(1,1)<-1** @depth 0: [0,0.494949], n=(50,50)
**GROW(0,1)<-1** @depth 1: [0,0.242424], n=(25,24)
**PRUNE(1,0)->1** @depth 1: [0,0.292929]
r=1000 corr=[0.00380686] [0] : n = 48 52
r=2000 corr=[0.00433483] [0] : n = 50 50

Obtaining samples (nn=99 predictive locations):
r=1000 corr=[0.00240425] [0.0652247] : mh=2 n = 48 52
r=2000 corr=[0.00723703] [0.0488049] : mh=2 n = 49 51
r=3000 corr=[0.00604759] [1.76362] : mh=2 n = 49 51
r=4000 corr=[0.00425486] [0] : mh=2 n = 51 49
r=5000 corr=[0.00351977] [1.00793] : mh=2 n = 48 52
Grow: 0.006006%, Prune: 0.00303%, Change: 0.5336%, Swap: 1%

finished repetition 1 0f 1
removed 2 leaves from the tree
```

The progress indicators show successful *grow* and *prune* operations, and every 1,000 rounds the partitions under the LLM show `corr=[0]`. Figure 8 shows the resulting posterior predictive surface and MAP partition ($\hat{\mathcal{T}}$).

3.3 Synthetic 2-d Exponential Data

The next example involves a two-dimensional input space in $[-2, 6] \times [-2, 6]$. The true response is given by

$$z(\mathbf{x}) = x_1 \exp(-x_1^2 - x_2^2). \tag{15}$$

A small amount of Gaussian noise (with sd = 0.001) is added. Besides its dimensionality, a key difference between this data set and the last one is that it is not defined using step functions; this smooth function does not have any artificial breaks between regions. The `tgp` package provides a function for data

```
> plot(sin.btgpllm, main = "treed GP LLM,")
```

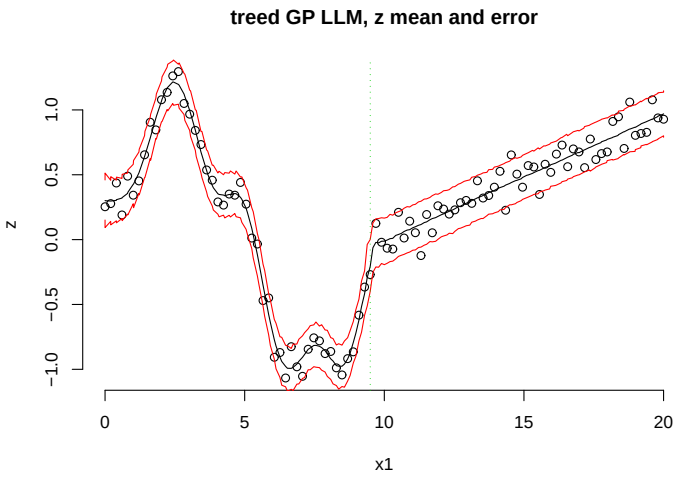


Figure 8: Posterior predictive distribution using `btgpllm` on synthetic sinusoidal data: mean and 90% credible interval, and MAP partition $\hat{\mathcal{T}}$.

subsampled from a grid of inputs and outputs described by (15) which concentrates inputs (`X`) more heavily in the first quadrant where the response is more interesting. Predictive locations (`XX`) are the remaining grid locations.

```
> exp2d.data <- exp2d.rand()
> X <- exp2d.data$X
> Z <- exp2d.data$Z
> XX <- exp2d.data$XX
```

CART is clearly just as inappropriate for this data as it was for the sinusoidal data in the previous section. However, a stationary GP fits this data just fine. After all, the process is quite well behaved. In two dimensions one has a choice between the isotropic and separable correlation functions. Separable is the default in the `tgp` package. For illustrative purposes here, I shall use the isotropic power family.

```
> exp.bgp <- bgp(X = X, Z = Z, XX = XX, corr = "exp")
```

Progress indicators are suppressed. Figure 9 shows the resulting posterior predictive surface under the GP in terms of means (*left*) and variances (*right*). The sampled locations (`X`) are shown as dots on the *right* image plot. Predictive locations (`XX`) are circles. Predictive uncertainty for the stationary GP model is highest where sampling is lowest, despite that the process is very uninteresting there. If any of the surface or perspective plots in the figure have white spaces, or holes, this is becuase of the `akima` bug mentioned in Section 1.1.1. This is not a bug in `tgp`.

```
> plot(exp.bgp, main = "GP,")
```

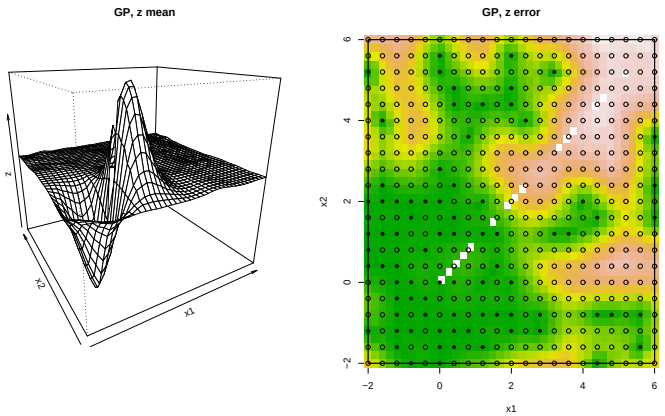


Figure 9: *Left:* posterior predictive mean using `bgp` on synthetic exponential data; *right* image plot of posterior predictive variance with data locations `X` (dots) and predictive locations `XX` (circles).

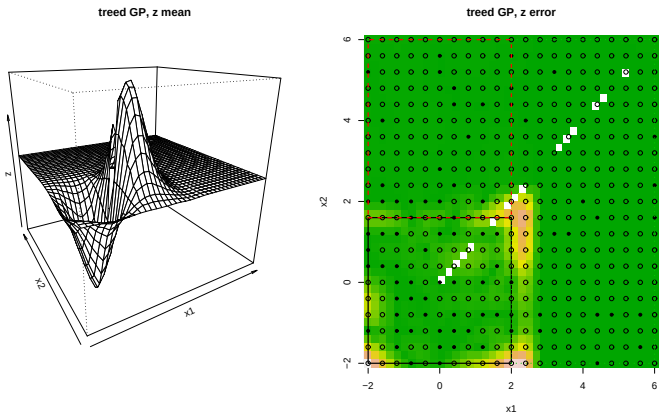
A treed GP seems more appropriate for this data. It can separate out the large uninteresting, essentially zero-response, part of the input space from the interesting part. The result is speedier inference, and region-specific estimates of predictive uncertainty. Chipman et al. recommend restarting the Markov chain a few times in order to better explore the marginal posterior for $\mathcal{T}$ [4]. This becomes more important for higher dimensional inputs, requiring deeper trees. The `tgp` default is `R = 1`, i.e., one chain with no restarts. Here two chains, with one restarts, are obtained using `R=2`.

```
> exp.btgp <- btgp(X = X, Z = Z, XX = XX, corr = "exp",
+     R = 2)
```

Figure 6 shows the resulting posterior predictive surface (*top*) and trees (*bottom*). Typical runs of the treed GP on this data find two, and if lucky three, partitions. As might be expected, jumping to the LLM for the uninteresting, zero-response, part of the input space can yield even further speedups [10].

```
> exp.btgpllm <- btgpllm(X = X, Z = Z, XX = XX, corr = "exp",
+     R = 2)

state = 201 588 370 ignored, using R RNG
n=80, d=2, nn=361, BTE=(2000,7000,2), R=2, linburn=0
predicting at data locations
correlation: isotropic power exponential
linear prior: flat
starting d = 0.5 0.5
starting nug=0.1, s2=1, tau2=1
starting beta = 0 0 0
```

```
> plot(exp.btgp, main = "treed GP,")
```




```
> tgp.trees(exp.btgp)
```

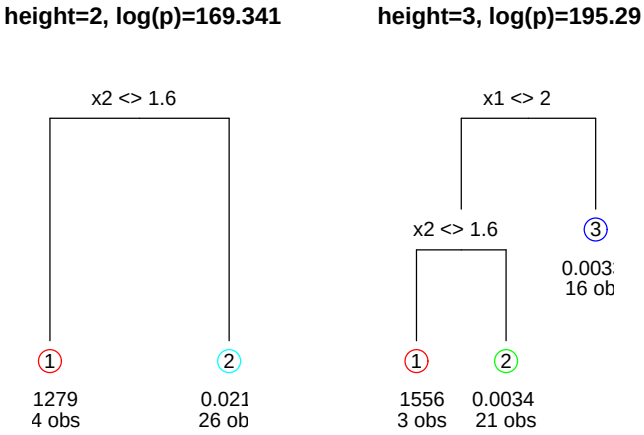


Figure 10: *Left:* posterior predictive mean using `btgp` on synthetic exponential data; *right* image plot of posterior predictive variance with data locations `X` (dots) and predictive locations `XX` (circles).

```
tree[alpha,beta]=[0.25,2], minpart=10
s2[a0,g0]=[5,10]
d[a,b][0,1]=[1,20],[10,10]
nug[a,b][0,1]=[1,1],[1,1]
gamlin = [10,0.2,0.7]
fixing d prior
```

```
fixing nug prior
s2 lambda[a0,g0]=[0.2,10]

burn in:
**GROW(0,1)<-0** @depth 0: [1,0.4], n=(50,30)
**PRUNE(0,1)->1** @depth 0: [1,0.35]
r=1000 corr=0.019141 : n = 80
r=2000 corr=0.0208016 : n = 80

Obtaining samples (nn=361 predictive locations):
**GROW(1,1)<-1** @depth 0: [1,0.5], n=(62,18)
r=1000 corr=0.0178545 0.904364 : mh=2 n = 62 18
**GROW(1,0)<-1** @depth 1: [0,0.5], n=(43,11)
r=2000 corr=0.0224447 1.1124 0.0198283 : mh=3 n = 43 11 26
r=3000 corr=0.0270723 0(1.66816) 0.061071 : mh=3 n = 50 14 16
r=4000 corr=0.0236126 0.199734 0(1.13711) : mh=3 n = 50 12 18
r=5000 corr=0.0194921 0(1.39435) 0(0.0761597) : mh=3 n = 50 15 15
Grow: 0.008264%, Prune: 0.004292%, Change: 0.2199%, Swap: 0.2895%

finished repetition 1 0f 2
removed 3 leaves from the tree

burn in:
**GROW(0,0)<-0** @depth 0: [0,0.5], n=(64,16)
**GROW(0,0)<-0** @depth 1: [1,0.5], n=(50,14)
**GROW(1,0)<-0** @depth 2: [0,0.1], n=(16,34)
**PRUNE(1,0)->1** @depth 2: [0,0.1]
r=1000 corr=0.0219221 0(0.791974) 0(0.764177) : mh=3 n = 50 12 18
r=2000 corr=0.0226169 0(1.60567) 0(0.656538) : mh=3 n = 50 12 18

Obtaining samples (nn=361 predictive locations):
r=1000 corr=0.0253895 0.0437966 1.78918 : mh=3 n = 43 21 16
r=2000 corr=0.0176578 0.0134262 0.714019 : mh=3 n = 43 21 16
r=3000 corr=0.021045 0.0128268 0.769922 : mh=3 n = 43 21 16
r=4000 corr=0.0215844 0.02203 0(1.34104) : mh=3 n = 43 21 16
r=5000 corr=0.0196245 0.00991285 0.0674148 : mh=3 n = 43 22 15
Grow: 0.008511%, Prune: 0.003431%, Change: 0.2139%, Swap: 0.2624%

finished repetition 2 0f 2
removed 3 leaves from the tree
```

Progress indicators show where the LLM (`corr=0(`d`)`) or the GP is active. Figure 11 show how similar the resulting posterior predictive surfaces are.

```
> plot(exp.btgpllm, main = "treed GP, LLM")
```

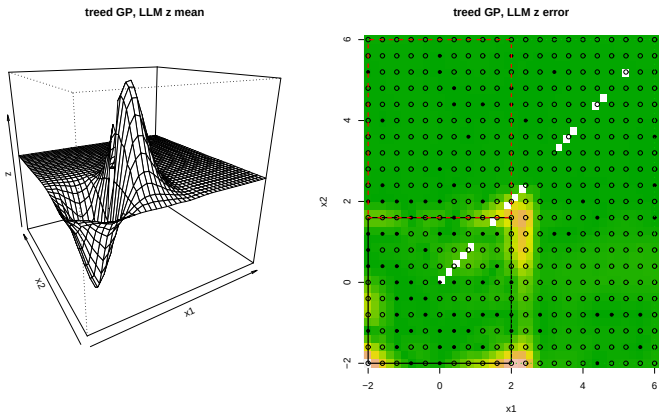


Figure 11: *Left:* posterior predictive mean using `btgpllm` on synthetic exponential data; *right* image plot of posterior predictive variance with data locations `X` (dots) and predictive locations `XX` (circles).

3.4 Motorcycle Accident Data

The Motorcycle Accident Dataset [21] is a classic nonstationary data set used in recent literature [18] to demonstrate the success of nonstationary models. The data consists of measurements of acceleration of the head of a motorcycle rider as a function of time in the first moments after an impact. In addition to being nonstationary, the data has input-dependent noise, which makes it useful for illustrating how the treed GP model handles this nuance. There are at least two, and perhaps three regions where the response exhibits different behavior both in terms of the correlation structure and noise level.

The data is included as part of the `MASS` library in `R`.

```
> library(MASS)
```

Figure 12 shows how a stationary GP is able to capture the nonlinearity in the response but fails to capture the input dependent noise, and increased smoothness (perhaps linearity) in parts of the input space.

```
> moto.bgp <- bgp(X = mcycle[, 1], Z = mcycle[, 2], m0r1 = TRUE)
```

Since the responses in this data have a wide range, it helps to translate and rescale them so that they have a mean of zero, and a range of one. The `m0r1` argument to `b*` and `tgp` functions automates this procedure. All progress indicators are surpressed for this example.

A Bayesian Linear CART model is able to capture the input dependent noise but fails to capture the waviness of the "whiplash"—center— segment of th the response.

```
> moto.btlm <- btlm(X = mcycle[, 1], Z = mcycle[, 2], m0r1 = TRUE)
```

```
> plot(moto.bgp, main = "GP,")
```

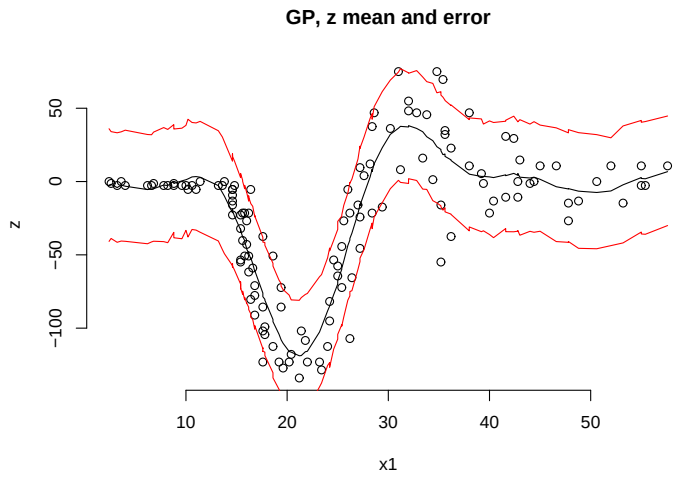


Figure 12: Posterior predictive distribution using `bgp` on the motorcycle accident data: mean and 90% credible interval

```
> plot(moto.btlm, main = "Bayesian CART,")
```

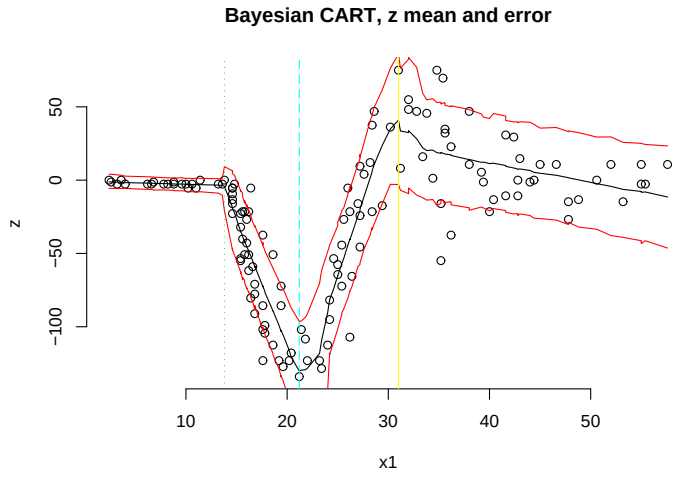


Figure 13: Posterior predictive distribution using `btlm` on the motorcycle accident data: mean and 90% credible interval

predictive surface and MAP partition ($\hat{T}$).

A treed GP model seems appropriate because it can model input dependent smoothness *and* noise. A treed GP LLM is probably most appropriate since the left-hand part of the input space is likely linear. One might further hypothesize

that the right-hand region is also linear, perhaps with the same mean as the left-hand region, only with higher noise. The `b*` and `tgp` functions can force an i.i.d. hierarchical linear model by setting `bprior=b0`. Moreover, instead of rescaling the responses with `m0r1`, one might try encoding a mixture prior for the nugget in order to explicitly model region-specific noise. This requires direct usage of `tgp`.

```
> p <- tgp.default.params(2)
> p$bprior <- "b0"
> p$nug.p <- c(1, 0.1, 10, 0.1)
> moto.tgp <- tgp(X = mcycle[, 1], Z = mcycle[, 2], params = p,
+     BTE = c(2000, 22000, 2))
```

The resulting posterior predictive surface is shown in *top* half of Figure 14. The *bottom* half of the figure shows the norm (difference) in predictive quantiles, clearly illustrating the treed GP's ability to capture input-specific noise in the posterior predictive distribution.

Other permutations of possible models, functions and arguments, for this data is contained in the `b*` and `tgp` examples sections of the respective `R` help files.

3.5 Friedman data

This Friedman data set is the first one of a suite that was used to illustrate MARS (Multivariate Adaptive Regression Splines) [9]. There are 10 covariates in the data ($\mathbf{x} = \{x_1, x_2, \ldots, x_{10}\}$). The function that describes the responses (Z), observed with standard Normal noise, has mean

$$E(Z|\mathbf{x}) = \mu = 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5, \qquad (16)$$

but depends only on $\{x_1, \ldots, x_5\}$, thus combining nonlinear, linear, and irrelevant effects. Comparisons are made on this data to results provided for several other models in recent literature. Chipman et al. [4] used this data to compare their linear CART algorithm to four other methods of varying parameterization: linear regression, greedy tree, MARS, and neural networks. The statistic they use for comparison is root mean-square error (RMSE)

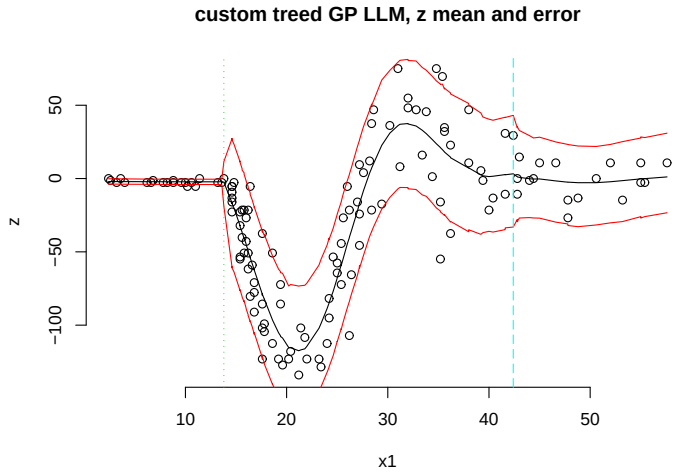
$$\text{MSE} = \textstyle\sum_{i=1}^{n}(\mu_i - \hat{z}_i)^2/n \qquad\qquad \text{RMSE} = \sqrt{\text{MSE}}$$

where $\hat{z}_i$ is the model-predicted response for input $\mathbf{x}_i$. The $\mathbf{x}$'s are randomly distributed on the unit interval.

Input data, responses, and predictive locations of size $N = 200$ and $N' = 1000$, respectively, can be obtained by a function included in the `tgp` package.

```
> f <- friedman.1.data(200)
> ff <- friedman.1.data(1000)
> X <- f[, 1:10]
> Z <- f$Y
> XX <- ff[, 1:10]
```

```
> plot(moto.tgp, main = "custom treed GP LLM,")
```




```
> main <- "quantile difference,"
> plot(moto.tgp$X[, 1], moto.tgp$Zp.q, type = "l", main = main)
```

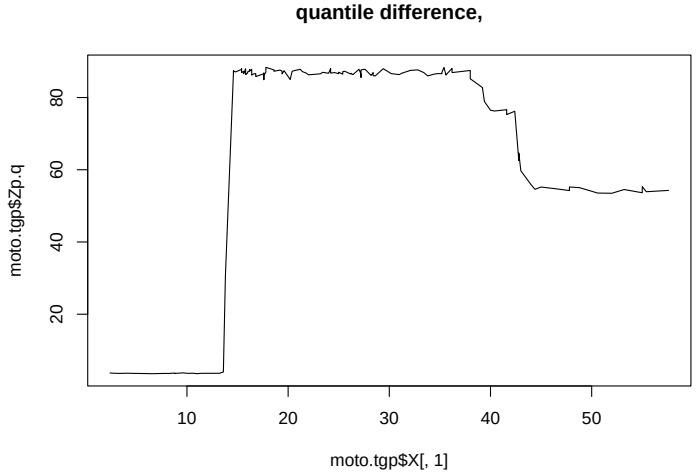


Figure 14: *top* Posterior predictive distribution using a custom parameterized `tgp` call on the motorcycle accident data: mean and 90% credible interval; *bottom* Quantile-norm (90%-5%) showing input-dependent noise.

This example compares Bayesian linear CART with Bayesian GP LLM (not treed), following the RMSE experiments of Chipman et al. It helps to scale the responses so that they have a mean of zero and a range of one. First, fit the Bayesian linear CART model, and obtain the RMSE.

```
> fr.btlm <- btlm(X = X, Z = Z, XX = XX, tree = c(0.95,
+     2, 10), m0r1 = TRUE)
> fr.btlm.mse <- sqrt(mean((fr.btlm$ZZ.mean - ff$Ytrue)^2))
> fr.btlm.mse
```

Next, fit the GP LLM, and obtain its RMSE.

```
> fr.bgpllm <- bgpllm(X = X, Z = Z, XX = XX, m0r1 = TRUE)
> fr.bgpllm.mse <- sqrt(mean((fr.bgpllm$ZZ.mean - ff$Ytrue)^2))
> fr.bgpllm.mse
```

So, the GP LLM is 4.834 times better than Bayesian linear CART on this data, in terms of RMSE (in terms of MSE the GP LLM is 2.199 times better). Watching the evolution of the Markov chain for the GP LLM (via the progress statements written to `stdout`, not shown because the would fit on the page), it is easy to see how the GP LLM quickly learns that $\mathbf{b} = (1, 1, 1, 0, 0, 0, 0, 0, 0, 0)$, and that $\beta_4 \approx 4$ and $\beta_5 \approx 10$—basically that only the first three inputs contribute nonlinearly, the fourth and fifth contribute linearly, and the remaining five not at all [10].

3.6 Adaptive Sampling

In this section, sequential design of experiments, a.k.a. *adaptive sampling*, is demonstrated on the exponential data of Section 3.3. Gathering, again, the data:

```
> exp2d.data <- exp2d.rand()
> X <- exp2d.data$X
> Z <- exp2d.data$Z
> Xcand <- exp2d.data$XX
```

Start by fitting a treed GP LLM model to the data, without prediction, in order to infer the MAP tree $\hat{\mathcal{T}}$.

```
> exp1 <- btgpllm(X = X, Z = Z, pred.n = FALSE, corr = "exp",
+     R = 2)
```

The trees are shown in Figure 15. Then, use the `tgp.design` function to create D-optimal candidate designs in each region of $\hat{\mathcal{T}}$.

```
> XX <- tgp.design(10, Xcand, exp1)

sequential treed D-Optimal design in 3 partitions
dopt.gp (1) choosing 2 new inputs from 63 candidates
dopt.gp (2) choosing 3 new inputs from 107 candidates
dopt.gp (3) choosing 6 new inputs from 191 candidates
```

Figure 16 shows the sampled `XX` locations (circles) amongst the input locations `X` (dots) and MAP partition ($\hat{\mathcal{T}}$). Notice how the candidates `XX` are spaced

```
> tgp.trees(exp1)
```

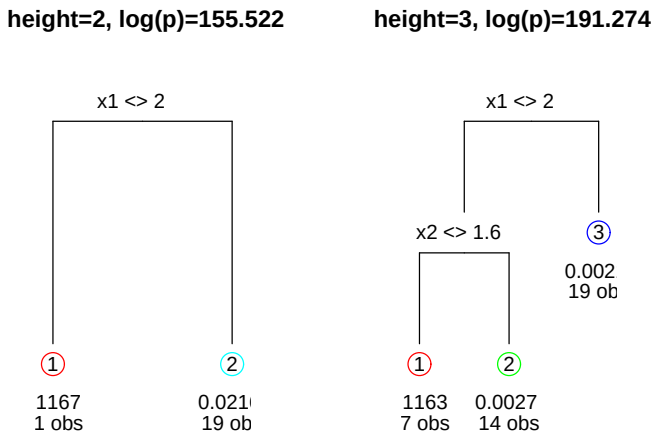


Figure 15: MAP trees of each height encountered in the Markov chain for the exponential data. $\hat{\mathcal{T}}$ is the one with the maximum $\log(p)$ above.

out relative to themselves, and relative to the inputs `X`, unless they are near partition boundaries. The placing of configurations near region boundaries is a symptom particular to D-optimal designs. This is desirable for experiments with `tgp` models, as model uncertainty is usually high there [2].

Figure 16 uses the `tgp.plot.parts.2d` function. Unfortunately, this function is not well documented in the current version of the `tgp` package. This should change in future versions.

Now, the idea is to fit the treed GP LLM model, again, in order to assess uncertainty in the predictive surface at those new candidate design points.

```
> exp1.btgpllm <- btgpllm(X = X, Z = Z, XX = XX, corr = "exp",
+     R = 2)
```

Figure 17 shows the posterior predictive surface. The error surface, on the *right*, summarizes posterior predictive uncertainty by a norm of quantiles. In accordance with the ALM algorithm, candidate locations `XX` with largest predictive error would be sampled (added into the design) next. These are most likely to be in the interesting region, i.e., the first quadrant. however, due to the random nature of this `Sweave` document, this is not always the case. Results depend heavily on the clumping of the original design in the un-interesting areas, and on the estimate of $\hat{\mathcal{T}}$.

Adaptive sampling via the ALC, or ego (or both) algorithms could proceed by setting any/all of the `Ds2x` or `ego` parameters to the `b*` and `tgp` to `TRUE`.

```
> plot(exp1$X, pch = 19, cex = 0.5)
> points(XX)
> tgp.plot.parts.2d(exp1$parts)
```

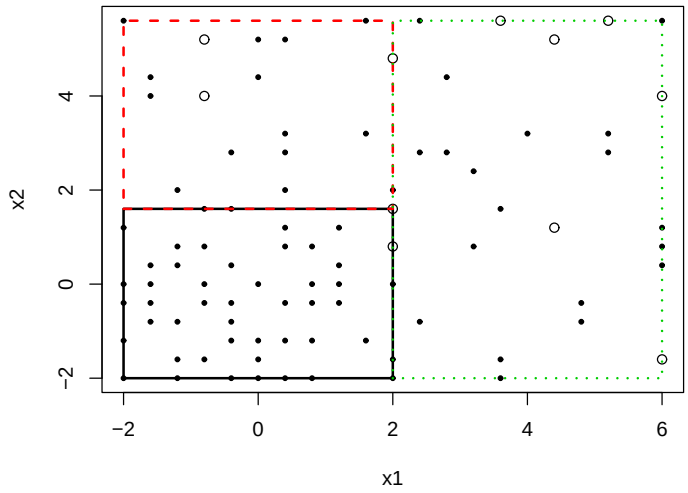


Figure 16: Treed D-optimal candidate locations `XX` (circles), input locations `X` (dots), and MAP tree $\hat{\mathcal{T}}$

```
> plot(exp1.btgpllm, main = "ALM for treed GP LLM,")
```

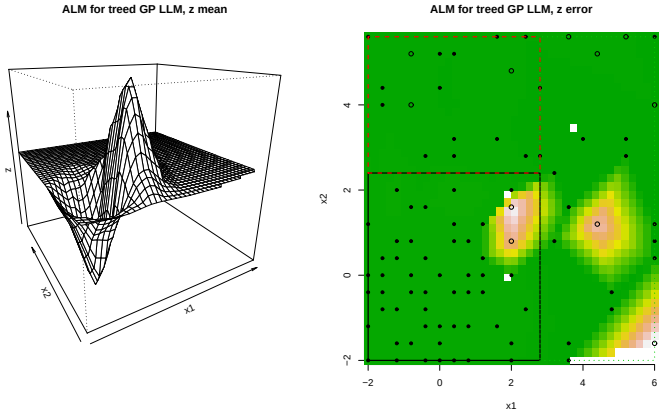


Figure 17: Treed D-optimal candidate locations `XX` (circles), input locations `X` (dots), and MAP tree $\hat{\mathcal{T}}$

A Linking to ATLAS

`ATLAS` is supported as an alternative to standard BLAS and LAPACK for fast, automatically tuned, linear algebra routines. There are three easy steps to

enable ATLAS support (assuming, of course, you have already installed it – http://math-atlas.sourceforge.net) which need to be done before you install the package from source. (Reverse the above instructions to disable ATLAS. Don't forget to re-install when you're done.)

1. Edit src/Makevars. Comment out the existing PKG_LIBS line, and replace it with:

   ```
   PGK_LIBS = -L/path/to/ATLAS/lib -llapack -lcblas -latlas
   ```

   You may need replace "-llapack -lcblas -latlas" with whatever ATLAS recommends for your OS. (see ATLAS README.) For example, if your ATLAS compilation included F77 support, you would might need to add "-lF77blas", of if you compiled with pthreads, you would might use "-llapack -lptcblas -lptf77blas -latlas".

2. Continue editing src/Makevars. Add:

   ```
   PKF_CFLAGS = -I/path/to/ATLAS/include
   ```

3. Edit src/linalg.h and commend out lines 40 & 41:

   ```
   /*#define FORTPACK
   #define FORTBLAS*/
   ```

In most cases, the ATLAS implementation is significantly faster than standard BLAS/Lapack. This is especially the case when compared to the BLAS/LAPACK that comes standard with R which is used in compiling R and R shared libraries (packages) as a last resort. Following the steps to install ATLAS for the tgp package in this case is *highly recommended*.

B Implementation

The treed GP model is coded in a mixture of C and C++: C++ for the tree data structure ($\mathcal{T}$) and C for the GP at each leaf of $\mathcal{T}$. The code has been tested on Unix (Solaris, Linux, FreeBSD, OSX) and Windows (2000, XP) platforms.

It is useful to first translate and re-scale the input data ($\mathbf{X}$) so that it lies in an $\Re^{m_X}$ dimensional unit cube. Doing this makes it easier to construct prior distributions for the width parameters to the correlation function $K(\cdot,\cdot)$ in particular. Proposals for all parameters which require MH sampling are taken from a uniform "sliding window" centered around the location of the last accepted setting. For example, a proposed a new nugget parameter g_ν to the correlation function $K(\cdot,\cdot)$ in region r_ν would go as

$$g_\nu^* \sim \text{Unif}\left(\frac{3}{4}g_\nu, \frac{4}{3}g_\nu\right).$$

Calculating the forward and backwards proposal probabilities for the MH acceptance ratio is straightforward.

After conditioning on the tree and parameters ($\{\mathcal{T}, \boldsymbol{\theta}\}$), prediction can be parallelized by using a producer/consumer model. This allows the use of the `PThreads` libraries in order to take advantage of multiple processors, and get speed-ups of at least a factor of two. The current `tgp` package is has contains a parallelized implementation, but it is not enabled by default. And unfortunately, no documentation yet exists to explain how to unleash this feature. This should be included in future versions (and help is available by personal correspondence). Parallel sampling of the posterior of $\boldsymbol{\theta}|\mathcal{T}$ for each of the $\{\theta_\nu\}_{\nu=1}^R$ is also possible. However, the speed-up in this second case is less impressive, and so is not supported by the current version available as an `R` package.

References

[1] L. Breiman, J. H. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, CA, 1984.

[2] K. Chaloner and I. Verdinelli. Bayesian experimental design, a review. *Statistical Science*, 10 No. 3:273–1304, 1995.

[3] H. A. Chipman, E. I. George, and R. E. McCulloch. Bayesian CART model search (with discussion). *Journal of the American Statistical Association*, 93:935–960, 1998.

[4] H. A. Chipman, E. I. George, and R. E. McCulloch. Bayesian treed models. *Machine Learning*, 48:303–324, 2002.

[5] D. A. Cohn. Neural network exploration using optimal experimental design. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6(9), pages 679–686. Morgan Kaufmann Publishers, 1996.

[6] N.A. Cressie. *Statistics for Spatial Data*. John Wiley and Sons, Inc., 1991.

[7] Dipak Dey, Peter Müeller, and Debajyoti Sinha. *Practical nonparametric and semiparametric Bayesian statistics*. Springer-Verlag New York, Inc., New York, NY, USA, 1998.

[8] Fields Development Team. fields: Tools for spatial data. National Center for Atmospheric Research, Boulder CO, 2004. URL: http://www.cgd.ucar.edu/Software/Fields.

[9] J. H. Friedman. Multivariate adaptive regression splines. *Annals of Statistics*, 19, No. 1:1–67, March 1991.

[10] R. B. Gramacy. *Bayesian Treed Gaussian Process Models*. PhD thesis, University of California, Santa Cruz, CA 95060, December 2005. Department of Applied Math & Statistics.

[11] R. B. Gramacy, Herbert K. H. Lee, and William Macready. Parameter space exploration with Gaussian process trees. In *ICML*, pages 353–360. Omnipress & ACM Digital Library, 2004.

[12] Robert B. Gramacy and Herbert K H. Lee. Gaussian processes and limiting linear models. Technical report, Dept. of Applied math & Statistics, University of California, Santa Cruz, 2005.

[13] D. Harrison and D. L. Rubinfeld. Hedonic housing prices and the demand for clean air. *Journal of Environmental Economics and Management*, 5:81–102, 1978.

[14] D. J. C. MacKay. Information-based objective functions for active data selection. *Neural Computation*, 4(4):589–603, 1992.

[15] G. Matheron. Principles of geostatistics. *Economic Geology*, 58:1246–1266, 1963.

[16] R. Neal. Monte carlo implementation of Gaussian process models for Bayesian regression and classification". Technical Report CRG–TR–97–2, Dept. of Computer Science, University of Toronto., 1997.

[17] R Development Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2004. ISBN 3-900051-00-3.

[18] C.E. Rasmussen and Z. Ghahramani. Infinite mixtures of Gaussian process experts. In *Advances in Neural Information Processing Systems*, volume 14, pages 881–888. MIT Press, 2002.

[19] T. J. Santner, B. J. Williams, and William I. Notz. *The Design and Analysis of Computer Experiments*. Springer-Verlag, New York, NY, 2003.

[20] S. Seo, M. Wallat, T. Graepel, and K. Obermayer. Gaussian process regression: Active data selection and test point rejection. In *Proceedings of the International Joint Conference on Neural Networks*, volume III, pages 241–246. IEEE, July 2000.

[21] B. W. Silverman. Some aspects of the spline smoothing approach to non-parametric curve fitting. *Journal of the Royal Statistical Society Series B*, 47:1–52, 1985.