# tgp v1.2: an R package for Bayesian nonstationary, semiparametric nonlinear regression and design by treed Gaussian process models

Robert B. Gramacy
Statistical Laboratory
University of Cambridge, United Kingdom
bobby@statslab.cam.ac.uk

November 27, 2006

**Abstract**

The tgp package for R [20] is a tool for fully Bayesian nonstationary, semiparametric nonlinear regression and design 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, the package supports the (sequential) design of experiments under these models paired with several objective criteria. 1-d and 2-d plotting, with higher dimension projection and slice capabilities, and tree drawing functions (requiring maptree and combinat libraries), are also provided for visualization of tgp-class output.

## Intended audience

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 the code quoted throughout is certified by R, and the Stangle command can be used to extract it.

Note that this tutorial was not meant to serve as an instruction manual. For more detailed documentation of the functions contained in the package, see the package help-manuals. At an R prompt, type help(package=tgp). PDF documentation is also available on the world-wide-web.

http://www.cran.r-project.org/doc/packages/tgp.pdf

| `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

The outline 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, compile–time support for `Pthreads` parallelization, the gathering of parameter traces, the verbosity of screen output, and some miscellaneous details of implementation.

## 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. 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 Section 2 and a demonstration is given in Section 3.6.

### 1.1 Bayesian regression models

The six regression models implemented in the 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 column of the table. The details of model specification and inference are 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 the core `C` code.

The `b*` functions are intended as the main interface, so little further attention to the `tgp` master function will be included here. The easiest way to see how the master `tgp` function implements one of the `b*` methods is to simply type the name of the function of interest into R. For example, to see the implementation of bgp, type:

```
> bgp
```

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 b* functions. Generic print, plot, and predict methods are defined for tgp-class objects. The plot and predict functions are discussed below. The print function simply provides a list of the names of the fields comprising a tgp-class object.

### 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.

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 image or perspective plots unless a 1-d projection argument is supplied. Data which are 3-d, or higher, require projection down to 2-d or 1-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 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 notes on 2-d plotting of tgp-class objects:

- 2-d plotting requires interpolation of the data onto a uniform grid. This is supported by the tgp package in two ways: (1) loess smoothing, and (2) the akima package, available from CRAN. The default is loess because it is more stable and does not require installing any further packages. When akima works it makes (in my opinion) smarter interpolations. However there are two bugs in the akima package, one malign and the other benign, which preclude it from the default position in tgp. The malign bug can cause a segmentation fault, and bring down the entire R session. The benign bug produces NA's when plotting data from a grid. For beautiful 2-d plots of gridded data I suggest exporting the tgp predictive output to a text file and using gnuplot's 2-d plotting features.

- 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 plotting is required. `tgp` methods have been used on data with input dimension as large as 15 [13], and were used in a sequential design and detailed analysis of some proprietary 3-d input and 6-d output data sampled using a NASA supercomputer [12].

- The `plot.tgp` function 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 diagrammatic 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.

#### 1.1.2 Prediction

Prediction, naturally, depends on fitted model parameters $\hat{\boldsymbol{\theta}}|\text{data}$, or Monte Carlo samples from the posterior distribution of $\boldsymbol{\theta}$ in a Bayesian analysis. Rather than saving samples from $\pi(\boldsymbol{\theta}|\text{data})$ for later prediction, usually requiring enormous amounts of storage, `tgp` samples the posterior predictive distribution in-line, as samples of $\boldsymbol{\theta}$ become available. [Section 2.1.4 and 2.2.1 outlines the prediction equations.] A `predict.tgp` function is provided should it be necessary to obtain predictions `after` the MCMC has finished. It gives the option of obtaining predictive means and variances from the MAP parameterization, $\hat{\boldsymbol{\theta}}$ maximizing $\pi(\boldsymbol{\theta}|Y)$, saved in the `tgp` class-output (from `b*`). In this way `predict.tgp` is similar to `predict.lm`, etc.. Samples can also be obtained from the MAP–parameterized predictive distributions via `predict.tgp`, or a re–initialization of the joint sampling of the posterior and posterior predictive distribution can commence starting from the $\hat{\boldsymbol{\theta}}$. The output of `predict.tgp` is also a `tgp` class object. Appendix B.3 illustrates how these features can be useful in the context of passing `tgp` model fits between collaborators. There are other miscellaneous demonstrations in Section 3.

#### 1.1.3 Speed

This is as good a place as any to make a disclaimer 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. Nor is inference for GP models, classical or Bayesian.

Great care has been taken to make the implementation of Bayesian inference for GP models as efficient as possible [see Appendix A]. However, inference for non-treed GPs can be computationally intense. Several features are implemented by the package which can help speed things up a bit. Direct support for `ATLAS` [26] is provided for fast linear algebra. Details on linking this package with `ATLAS` is contained in Appendix C.1. Parallelization of prediction and inference is supported by a producer/consumer model implemented with `Pthreads`. Appendix C.2 shows how to activate this feature, as it is not turned on by default. An argument called `linburn` is made available in tree class (T) `b*` functions in Table 1. When `linburn = TRUE`, the Markov chain is initialized with a run of the Bayesian linear CART algorithm [5] before burn-in in order to pre-partition the input space using linear models. Finally, thinning of the posterior predictive samples obtained by the Markov chain can also help speed things up. This is facilitated by the `E`-part of the `BTE` argument to `b*` functions.

## 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. Nevertheless, options and functions are provided 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 include `Ds2x` and `improv`. Both are booleans, i.e., should be set to `TRUE` or `FALSE` (the default for both is `FALSE`). `TRUE` booleans cause the `tgp`-class output list to contain vectors of the similar names which contain statistics that can be used toward adaptive sampling. When `Ds2x = TRUE` then $\Delta\sigma^2(\tilde{\mathbf{x}})$ statistic is computed at each $\tilde{\mathbf{x}} \in$ `XX`, in accordance the ALC (Active Learning–Cohn) algorithm [6]. Likewise, when `improv = TRUE`, statistics are computed in order to asses the expected information gain, a.k.a, expected improvement (EI), for each $\tilde{\mathbf{x}} \in$ `XX` about the global minimum [15] . The ALM (Active Learning–Mackay) algorithm [16] 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 on the ALM, ALC, and EI algorithms are provided in Section 2.

Calculation of EI statistics is considered to be “beta” functionality in this version of the `tgp` package. It has not been adequately tested, and its implementation is likely to change substantially in future versions of the package. In particular, the search good candidate configurations $\tilde{\mathbf{X}}$ for evaluating the EI statistic, analogous to the branch and bound Expected Global Optimization (EGO) algorithm [15], is still an open research question.

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, EI) can gather statistics. The `dopt.gp` function generates $D$–optimal candidates for a stationary GP. 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, including the original paper on Bayesian treed Gaussian process models [13], and an application paper on adaptively designing supercomputer experiments [12].

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 $D = \{\mathbf{X}, \mathbf{Z}\}$ where $m_X$ is the number of covariates (columns) of the design matrix $\mathbf{X}$ and $m \equiv m_X + 1$.

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

where $\mathbf{F} = (\mathbf{1}, \mathbf{X})$, and $\mathbf{W}$ is a $m \times m$ 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, $g$ is the *nugget*, and $K^*$ is a *true* correlation representative from a parametric family. The isotropic Matérn family is also implemented in the current version as "beta" functionality.

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 [13].

#### 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 [17, 7] and sometimes as *jitter* in the Machine Learning literature [18]. 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 $\eta(\cdot)$ represents i.i.d. 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 $\mathbf{x}_k$: $||\mathbf{x}_j - \mathbf{x}_k||$.

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

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 *separable* 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 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 Matérn Family

Another popular set of correlation functions is the Matérn family, due to many nice properties [25, 19]. Correlations in this family are isotropic, and have the form:

$$K(\mathbf{x}_j, \mathbf{x}_k|\nu, \phi, \alpha) = \frac{\pi^{1/2}\phi}{2^{\nu-1}\Gamma(\nu + 1/2)\alpha^{2\nu}}(\alpha||\mathbf{x}_j - \mathbf{x}_k||)^{\nu}\mathcal{K}_{\nu}(\alpha||\mathbf{x}_j - \mathbf{x}_k||) \tag{5}$$

where $\mathcal{K}_\nu$ is a modified Bessel function of the second kind [1]. This family of correlation functions are obtained from spectral densities of the form $f(\omega) = \phi(\alpha^2 + \omega^2)^{-\nu-1/2}$. Since the resulting process can shown to be $\lceil\nu\rceil - 1$ times differentiable, $\nu$ can be thought of as a smoothness parameter. The ability to specify smoothness is a significant feature of the Matérn family, especially in comparison to the power exponential family which is either nowhere differentiable ($0 < p_0 < 2$) or infinitely differentiable ($p_0 = 2$).

Separable parameterizations of the Matérn family also exist, but the current version of `tgp` supports only the isotropic parameterization, for fixed $\nu$. Future versions will allow $\nu$ to be estimated, and support both isotropic and separable parameterizations.

### 2.1.4 Prediction and Adaptive Sampling

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

$$\hat{z}(\mathbf{x}) = \mathbf{f}^\top(\mathbf{x})\tilde{\boldsymbol{\beta}} + \mathbf{k}(\mathbf{x})^\top \mathbf{K}^{-1}(\mathbf{Z} - \mathbf{F}\tilde{\boldsymbol{\beta}}), \tag{6}$$

$$\hat{\sigma}^2(\mathbf{x}) = \sigma^2[\kappa(\mathbf{x},\mathbf{x}) - \mathbf{q}^\top(\mathbf{x})\mathbf{C}^{-1}\mathbf{q}(\mathbf{x})], \tag{7}$$

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

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

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 [17].

The ALM algorithm [16] is implemented with MCMC by computing the norm (or width) of predictive quantiles obtained by samples from the Normal distribution given above. The ALC algorithm [6] 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 $\tilde{\mathbf{Y}}$):

$$\begin{aligned}\Delta\hat{\sigma}^2(\tilde{\mathbf{x}}) &= \frac{1}{|\tilde{\mathbf{Y}}|}\sum_{\mathbf{y}\in\tilde{\mathbf{Y}}} \Delta\hat{\sigma}^2_{\mathbf{y}}(\tilde{\mathbf{x}}) = \frac{1}{|\tilde{\mathbf{Y}}|}\sum_{\mathbf{y}\in\tilde{\mathbf{Y}}} \hat{\sigma}^2_{\mathbf{y}} - \hat{\sigma}^2_{\mathbf{y}}(\tilde{\mathbf{x}}) \\ &= \frac{1}{|\tilde{\mathbf{Y}}|}\sum_{\mathbf{y}\in\tilde{\mathbf{Y}}} \frac{\sigma^2\left[\mathbf{q}_N^\top(\mathbf{y})\mathbf{C}_N^{-1}\mathbf{q}_N(\tilde{\mathbf{x}}) - \kappa(\tilde{\mathbf{x}},\mathbf{y})\right]^2}{\kappa(\tilde{\mathbf{x}},\tilde{\mathbf{x}}) - \mathbf{q}_N^\top(\tilde{\mathbf{x}})\mathbf{C}_N^{-1}\mathbf{q}_N(\tilde{\mathbf{x}})},\end{aligned} \tag{8}$$

which is easily computed using MCMC methods [12]. 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 [15] is based on a statistic which captures the expected improvement (EI) in the model about its ability to predict the spatial location of the global minimum. If $f_{\min}$ is the current minimum, e.g., $f_{\min} = \min\{z_1, \ldots, z_N\}$, then the EI at the point $\tilde{\mathbf{x}}$ can reasonably be encoded as

$$E[I(\tilde{\mathbf{x}})] = E[\max(f_{\min} - Z(\tilde{\mathbf{x}}), 0)], \tag{9}$$

which can be shown to work out to be

$$E[I(\tilde{\mathbf{x}})] = (f_{\min} - \hat{z}(\tilde{\mathbf{x}}))\Phi\left(\frac{f_{\min} - \hat{z}(\tilde{\mathbf{x}})}{\hat{\sigma}(\tilde{\mathbf{x}})}\right) + \hat{\sigma}(\tilde{\mathbf{x}})\phi\left(\frac{f_{\min} - \hat{z}(\tilde{\mathbf{x}})}{\hat{\sigma}(\tilde{\mathbf{x}})}\right) \tag{10}$$

where $\hat{z}$ and $\hat{\sigma} = \sqrt{\hat{\sigma}^2}$ are taken from the equations for the posterior predictive distribution (6). $\Phi$ and $\phi$ are the standard Normal cumulative distribution and probability density functions, respectively.

The `tgp` package computes the expectation in (9) via MCMC samples from the improvement $\max\{f_{\min} - Z(\tilde{\mathbf{x}}), 0\}$ at locations $\tilde{\mathbf{x}} \in \tilde{\mathbf{X}}$. However, the method uses $\min_{i=1}^n\{Z_{\mathbf{x}_i}\}$, a sample from the first order statistic of the posterior predictive distribution at the inputs $\mathbf{x} \in \mathbf{X}$, in place of $f_{\min}$. An exception is when the argument `pred.n = FALSE` is provided instructing `tgp` not to sample from the posterior predictive distribution of the input locations $\mathbf{X}$. In this case, the original closed form EI formula (10) is used.

## 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 [13]. 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 $i^{\text{th}}$ 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 $d_i$: $p(b_i, d_i) = p(b_i|d_i)p(d_i)$.

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

$$p_{\gamma,\theta_1,\theta_2}(b_i = 0|d_i) = \theta_1 + (\theta_2 - \theta_1)/(1 + \exp\{-\gamma(d_i - 0.5)\}) \tag{11}$$

with $0 < \gamma$ and $0 \leq \theta_1 \leq \theta_2 < 1$, can encode such a preference by calling for the exclusion of dimensions $i$ 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 $i^{\text{th}}$ dimension. Accordingly, $\theta_1$ and $\theta_2$ represent minimum and maximum probabilities of jumping to the LLM, while $\gamma$ governs the rate at which $p(b_i =$

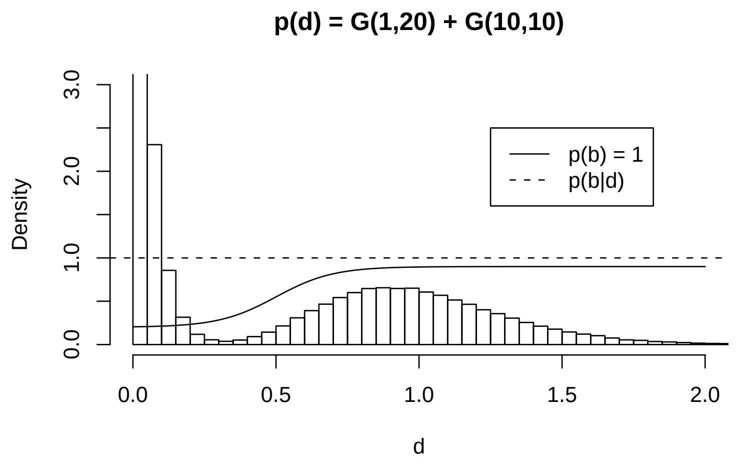


Figure 1: Prior distribution for the boolean ($b$) superimposed on $p(d)$.

$0|d_i)$ grows to $\theta_2$ 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,

$$p(d) = [G(d|\alpha = 1, \beta = 20) + G(d|\alpha = 10, \beta = 10)]/2, \tag{12}$$

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 $\theta_2$ 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

$$p(\text{linear model}) = \prod_{i=1}^{m_X} p(b_i = 0|d_i) = \prod_{i=1}^{m_X} \left[\theta_1 + \frac{\theta_2 - \theta_1}{1 + \exp\{-\gamma(d_i - 0.5)\}}\right]. \tag{13}$$

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 $\mathbf{x}'$. Such models are commonplace in the statistics community [8]. Traditionally, $\mathbf{x}'$ is determined and fixed *a' priori*. The separable boolean prior (11) implements an adaptively semiparametric process where the subset $\mathbf{x}' = \{x_i : b_i = 1, i = 1, \ldots, m_X\}$ is given a prior distribution, instead of being fixed.

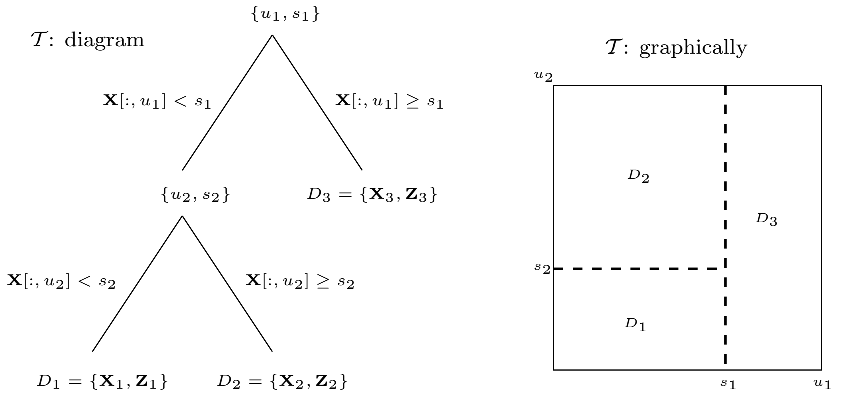

$\mathcal{T}$: diagram
$\{u_1, s_1\}$
$\mathbf{X}[:, u_1] < s_1$
$\mathbf{X}[:, u_1] \geq s_1$
$\{u_2, s_2\}$
$D_3 = \{\mathbf{X}_3, \mathbf{Z}_3\}$
$\mathbf{X}[:, u_2] < s_2$
$\mathbf{X}[:, u_2] \geq s_2$
$D_1 = \{\mathbf{X}_1, \mathbf{Z}_1\}$
$D_2 = \{\mathbf{X}_2, \mathbf{Z}_2\}$
$\mathcal{T}$: graphically
$u_2$
$D_2$
$D_3$
$s_2$
$D_1$
$s_1$
$u_1$

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

### 2.2.1 Prediction and Adaptive Sampling under LLM

Prediction under the limiting GP model is a simplification of (6) when it is known that $\mathbf{K} = (1+g)\mathbf{I}$. It can be shown [13] 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 (6) with $\mathbf{K} = \mathbf{I}(1+g)$ because an $m \times m$ inversion is faster than an $n \times n$ one.

Applying the ALC algorithm under the LLM also offers computational savings. intense compared to ALC under a full GP. Starting with the predictive variance given in (6), the expected reduction in variance under the LM is [12]

$$\Delta\hat{\sigma}^2_{\mathbf{y}}(\mathbf{x}) = \frac{\sigma^2[\mathbf{f}^\top(\mathbf{y})\mathbf{V}_{\tilde{\beta}_N}\mathbf{f}(\mathbf{x})]^2}{1+g+\mathbf{f}^\top(\mathbf{x})\mathbf{V}_{\tilde{\beta}_N}\mathbf{f}(\mathbf{x})} \tag{14}$$

which is similarly preferred over (8) with $\mathbf{K} = \mathbf{I}(1+g)$.

The statistic for expected improvement (EI; about the minimum) is the same under the LLM as (10) 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 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) [2]. 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 trees implemented in the `tgp` package follow Chipman et al. [4] 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, $\eta$, being split with probability $p_{\mathrm{SPLIT}}(\eta, \mathcal{T}) = a(1 + q_\eta)^{-b}$ where $q_\eta$ is the depth of $\eta$ 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. [5], the `tgp` package implements a stationary GP with linear trend, or GP LLM, independently within each of the regions depicted by a tree $\mathcal{T}$ [13]. Integrating out dependence on $\mathcal{T}$ is accomplished by reversible-jump MCMC (RJ-MCMC) via tree operations *grow, prune, change*, and *swap* [4]. To keep things simple, proposals for new parameters—via an increase in the number of partitions (through a *grow*)—are drawn from their priors[1], 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 is augmented with a *rotate* option to improve mixing of the Markov chain [13].

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, fully Bayesian model averaging 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* [22]. 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 GPs with correlation matrix $\mathbf{K}$, this is equivalent to maximizing $\det(\mathbf{K})$. Subsequently chosen input configurations are called $D-$optimal designs. Choosing quadratic

[1]Proposed *grows* are the *only* place where the priors (for $d$, $g$ and $\tau^2$ parameters; the others can be integrated out) are used for MH–style proposals. All other MH proposals are "random–walk" as described in Appendix A.

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 [3].

Other approaches used by the statistics community include space-filling designs: e.g. max-min distance and Latin Hypercube (LH) designs [22]. The FIELDS package [9], 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 should be considered for adding into the design. The Active Learning–MacKay (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 [16]. An alternative algorithm is to select $\tilde{\mathbf{x}}$ minimizing the resulting expected squared error averaged over the input space [6], called ALC for Active Learning–Cohn. Seo et al. [23] provide a comparison between ALC and ALM using standard GPs. The EI [15] 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 [12]. 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 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 [22]. 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. This strategy tends to work well with ALM and ALC, however it is less than ideal for EI as will be illustrated in Section 3.6. Adaptive sampling from EI (with `tgp`) is still an open area of research.

## 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, the code for the exponential data of Section 3.3 can be extracted 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 invoke 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.

Each subsection (or subsection of the appendix) starts by seeding the random number generator with `set.seed(0)`. This is done to make the results and analyses reproducible within this document, and in demo form. I recommend you try these examples with different seeds and see what happens. Usually the results will be similar, but sometimes (especially when the data (`X, Z`) is generated randomly) they may be quite different.

Other successful uses of the methods in this package include applications to the Boston housing data [14, 13], and designing an experiment for a reusable NASA launch vehicle [11, 12] called the Langely glide-back booster (LGBB).

### 3.1 1-d Linear data

Consider data sampled from a linear model.

$$z_i = 1 + 2x_i + \epsilon, \quad \text{where} \quad \epsilon_i \overset{\text{iid}}{\sim} N(0, 0.25^2) \tag{15}$$

The following `R` code takes a sample $\{\mathbf{X}, \mathbf{Z}\}$ of size $N = 50$ from (15). 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)
```

```
burn in:
r=1000 d=[0]; n=50

Sampling @ nn=99 pred locs:
r=1000 d=[0]; mh=1 n=50
r=2000 d=[0]; mh=1 n=50
r=3000 d=[0]; mh=1 n=50
```

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

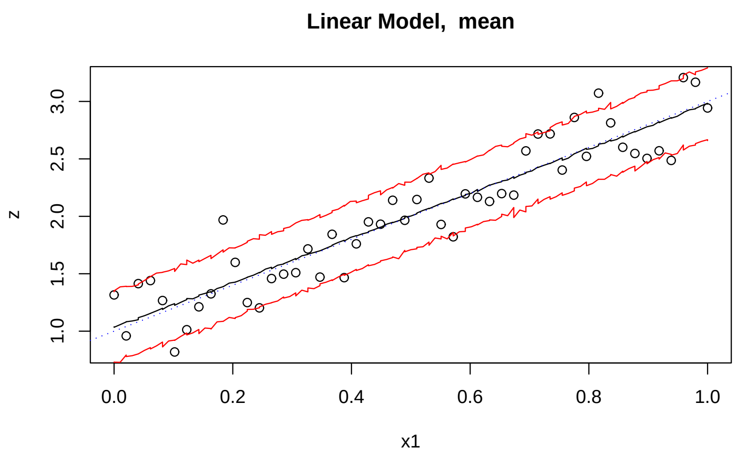


Figure 3: Posterior predictive distribution using `blm` on synthetic linear data: mean and 90% credible interval. The actual generating lines are shown as blue-dotted.

MCMC progress indicators are echoed every 1,000 rounds. The linear model is indicated by `d=[0]`. For `btlm` the MCMC progress indicators are boring, but we will see more interesting ones later. In terminal versions, e.g. `Unix`, the progress indicators can give a sense of when the code will finish. GUI versions of `R`—`Windows` or `MacOS X`—can buffer `stdout`, rendering this feature essentially useless as a real–time indicator of progress. Progress indicators can be turned off by providing the argument `verb=0`. Further explanation on the verbosity of screen output and interpretations is provided in Appendix B.2.

The generic `plot` method can be used to visualize the fitted posterior predictive surface (with option `layout = 'surf'`) in terms of means and credible intervals. Figure 3 shows how to do it, and what you get. The default option `layout = 'both'` shows both a predictive surface and error (or uncertainty) plot, side by side. The error plot can be obtained alone via `layout = 'as'`. Examples of these layouts appear later.

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

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

burn in:
r=1000 d=[0]; n=50

Sampling @ nn=99 pred locs:
r=1000 d=[0]; mh=1 n=50
r=2000 d=[0]; mh=1 n=50
r=3000 d=[0]; mh=1 n=50
```

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

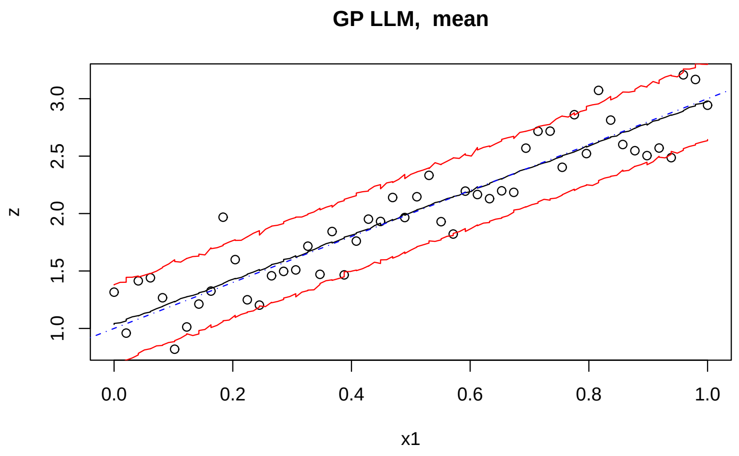


Figure 4: Posterior predictive distribution using `bgpllm` on synthetic linear data: mean and 90% credible interval. The actual generating lines are shown as blue-dotted.

Whenever the progress indicators show `d=[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 $N < 20$.

To see the proportion of time the Markov chain spent in the LLM requires the gathering of traces (Appendix B.1). For example

```
> lin.gpllm.tr <- bgpllm(X = X, XX = 0.5, Z = Z, pred.n = FALSE,
+     trace = TRUE, verb = 0)
> mla <- mean(lin.gpllm.tr$trace$linarea$la)
> mla

[1] 0.96
```

shows that the average area under the LLM is 0.96. Progress indicators are suppressed with `verb=0`. Alternatively, the probability that input location `xx` = 0.5 is under the LLM is given by

```
> 1 - mean(lin.gpllm.tr$trace$XX[[1]]$b1)

[1] 0.96
```

This is the same value as the area under the LLM since the process is stationary (i.e., there is no treed partitioning).

## 3.2 1-d Synthetic Sine Data

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

$$z(x) = \begin{cases} \sin\left(\frac{\pi x}{5}\right) + \frac{1}{5}\cos\left(\frac{4\pi x}{5}\right) & x < 10 \\ x/10 - 1 & \text{otherwise} \end{cases} \tag{16}$$

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. Some evenly spaced predictive locations `XX` are also created.

```
> 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)
```

By design, the data is clearly nonstationary. Perhaps not knowing this, a good first model choice for this data might be a GP.

```
> sin.bgp <- bgp(X = X, Z = Z, XX = XX, verb = 0)
```

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 because the data comes from a process that is nonstationary.

So one might consider a Bayesian treed linear model (LM) instead.

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

burn in:
**GROW** @depth 0: [0,0.424242], n=(43,57)
**GROW** @depth 1: [0,0.252525], n=(26,19)
**GROW** @depth 2: [0,0.131313], n=(14,13)
r=1000 d=[0] [0] [0] [0]; n=(11,19,17,53)
```

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

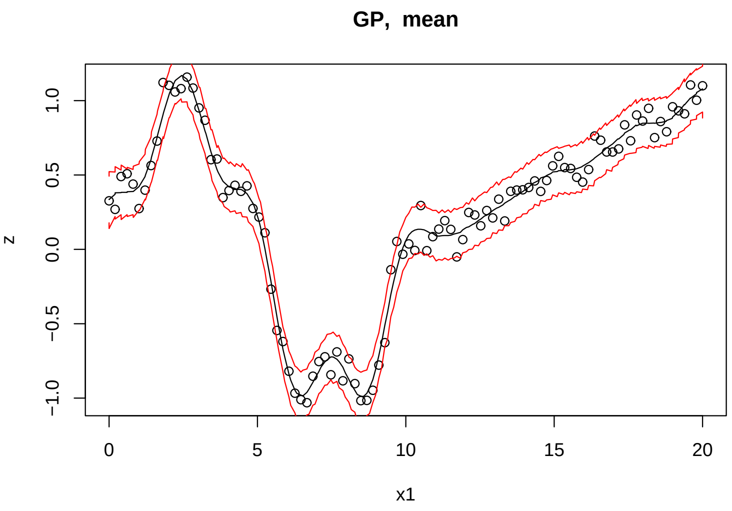


Figure 5: Posterior predictive distribution using `bgp` on synthetic sinusoidal data: mean and 90% credible interval

```
r=2000 d=[0] [0] [0] [0]; n=(11,17,19,53)

Sampling @ nn=99 pred locs:
r=1000 d=[0] [0] [0] [0]; mh=3 n=(15,14,18,53)
r=2000 d=[0] [0] [0] [0]; mh=4 n=(14,14,19,53)
r=3000 d=[0] [0] [0] [0]; mh=4 n=(12,16,19,53)
r=4000 d=[0] [0] [0] [0]; mh=4 n=(13,16,18,53)
r=5000 d=[0] [0] [0] [0]; mh=4 n=(13,15,19,53)
Grow: 0.8403%, Prune: 0%, Change: 36.3%, Swap: 84%
```

MCMC progress indicators show successful *grow* and *prune* operations as they happen, and region sizes $n$ every 1,000 rounds. Specifying `verb=3`, or higher will show echo more successful tree operations, i.e., *change*, *swap*, and *rotate*.

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 treed LM 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.

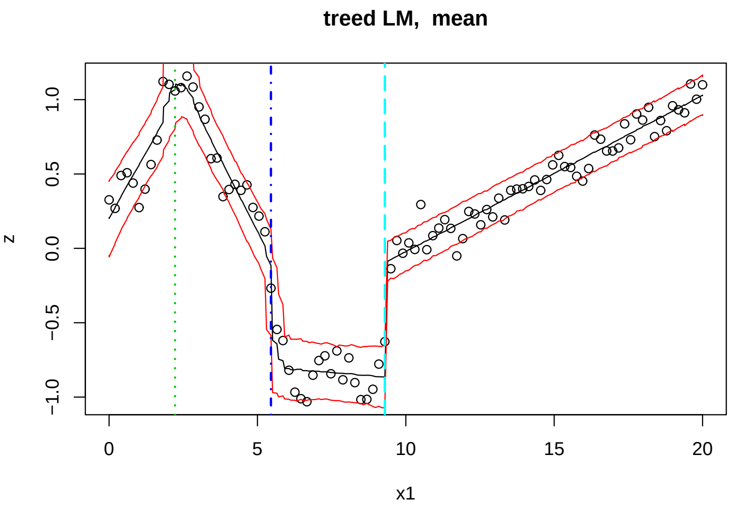
The ideal model for this data is the Bayesian treed GP because it can be both smooth and wiggly.

```
> sin.btgp <- btgp(X = X, Z = Z, XX = XX, verb = 0)
```

Figure 7 shows the resulting posterior predictive surface (*top*) and MAP $\hat{\mathcal{T}}$ with height=2.

Finally, speedups can be obtained if the GP is allowed to jump to the LLM [13], since half of the response surface is *very* smooth, or linear. This is not

```
> plot(sin.btlm, main = "treed LM,", layout = "surf")
```



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

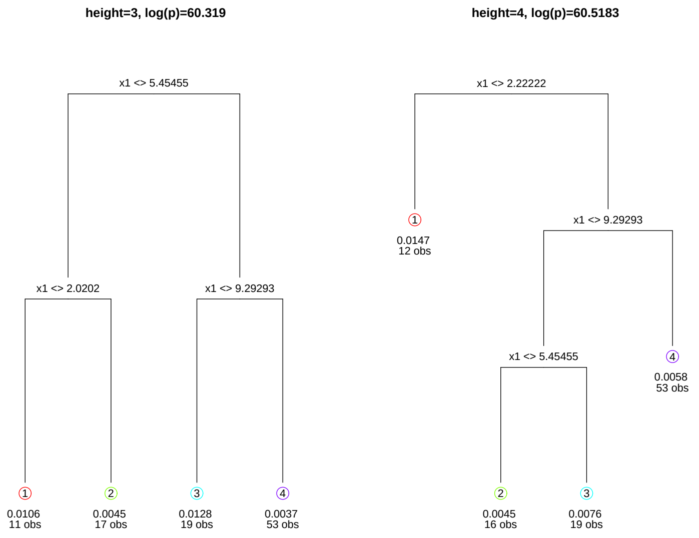


Figure 6: *Top:* Posterior predictive distribution using `btlm` on synthetic sinusoidal data: mean and 90% credible interval, and MAP partition ($\hat{\mathcal{T}}$); *Bottom* MAP trees for each height encountered in the Markov chain showing $\hat{\sigma}^2$ and the number of observation $n$, at each leaf.

shown here since the results are very similar to those above, replacing `btgp` with `btgpllm`. The example in the next subsection offers a comparison for 2-d data.

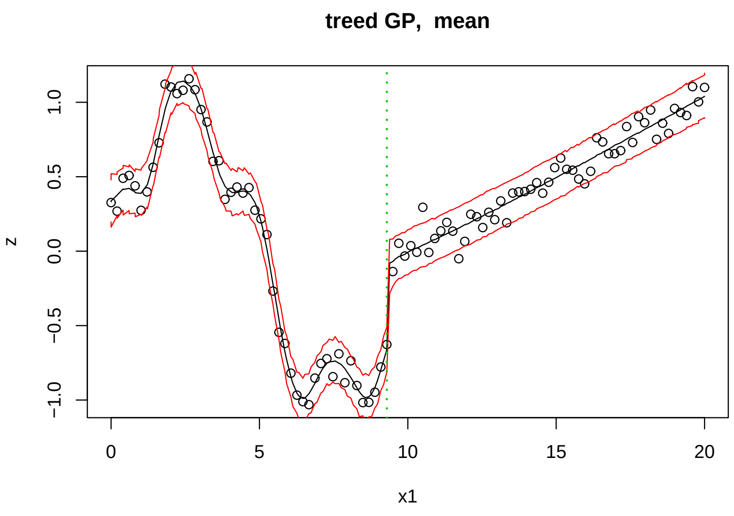


Figure 7: Posterior predictive distribution using `btgp` on synthetic sinusoidal data: mean and 90% credible interval, and MAP partition ($\hat{\mathcal{T}}$)

## 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

$$z(\mathbf{x}) = x_1 \exp(-x_1^2 - x_2^2). \tag{17}$$

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 subsampled from a grid of inputs and outputs described by (17) 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
```

The treed LM 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",
+     verb = 0)
```

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

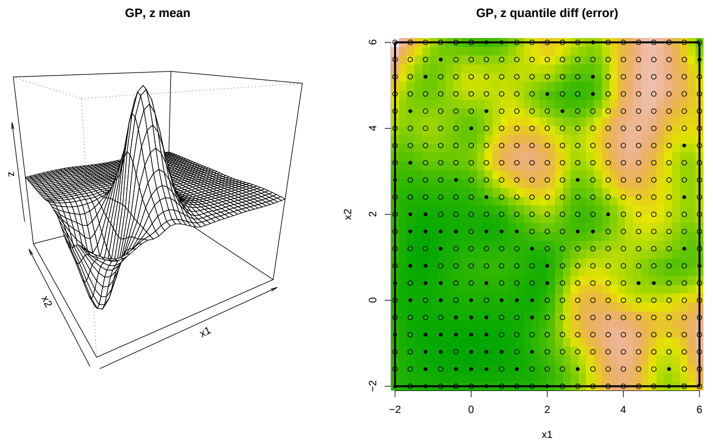


Figure 8: *Left:* posterior predictive mean using `bgp` on synthetic exponential data; *right* image plot of posterior predictive variance with data locations `X` (dots) and predictive locations `XX` (circles).

Progress indicators are suppressed. Figure 8 shows the resulting posterior predictive surface under the GP in terms of means (*left*) and variances (*right*) in the default layout. 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.

A treed GP seems more appropriate for this data. It can separate out the large uninteresting part of the input space from the interesting part. The result is speedier inference and region-specific estimates of predictive uncertainty.

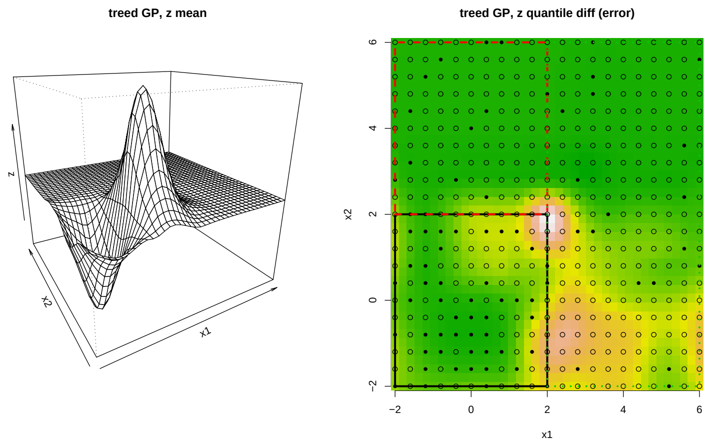
```
> exp.btgp <- btgp(X = X, Z = Z, XX = XX, corr = "exp",
+     verb = 0)
```

Figure 9 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 [13]. Also, Chipman et al. recommend restarting the Markov chain a few times in order to better explore the marginal posterior for $\mathcal{T}$ [5]. This can be important for higher dimensional inputs requiring deeper trees. The `tgp` default is `R = 1`, i.e., one chain with no restarts. Here two chains—one restart—are obtained using `R = 2`.

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

burn in:
**GROW** @depth 0: [0,0.45], n=(58,22)
```

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



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

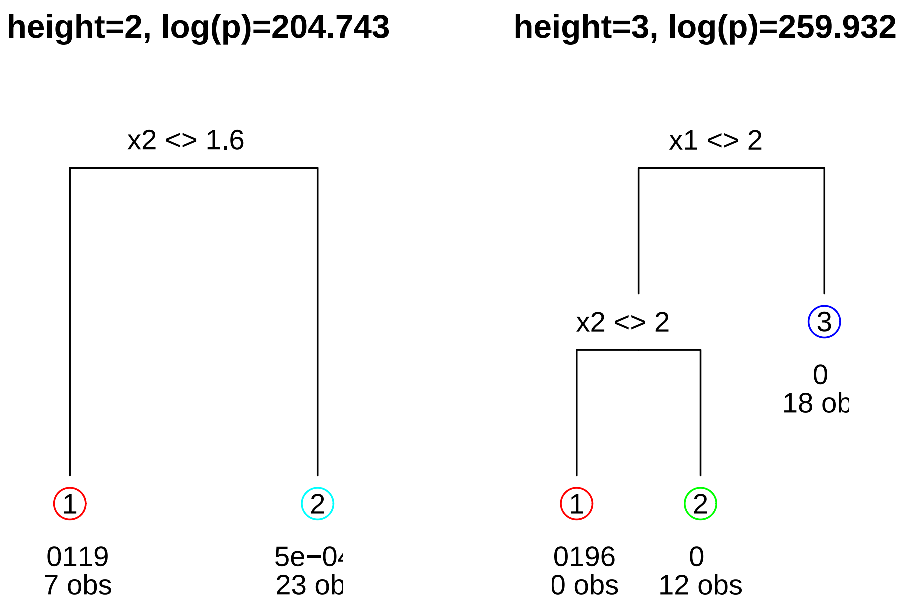


Figure 9: *Top-Left:* posterior predictive mean using `btgp` on synthetic exponential data; *top-right* image plot of posterior predictive variance with data locations `X` (dots) and predictive locations `XX` (circles). `Bottom:` MAP trees of each height encountered in the Markov chain with $\hat{\sigma}^2$ and the number of observations $n$ at the leaves.

```
**GROW** @depth 1: [1,0.45], n=(48,14)
r=1000 d=0.0218383 0(0.743906) 0(1.25691); n=(50,10,20)
r=2000 d=0.0222512 0.00696964 0(1.01303); n=(48,15,17)
```

```
Sampling @ nn=361 pred locs:
r=1000 d=0.0233355 0(0.0162573) 0(1.05867); mh=3 n=(50,10,20)
r=2000 d=0.0201525 0.0323228 0.735674; mh=3 n=(48,15,17)
r=3000 d=0.0212731 0.00712512 0.214455; mh=3 n=(48,14,18)
r=4000 d=0.0235209 0(0.696318) 0(0.0374696); mh=3 n=(50,10,20)
r=5000 d=0.019617 0(0.840593) 0.0798781; mh=3 n=(50,12,18)
Grow: 0.5882%, Prune: 0%, Change: 36.02%, Swap: 44.1%
finished repetition 1 of 2
removed 3 leaves from the tree

burn in:
**GROW** @depth 0: [1,0.5], n=(60,20)
**GROW** @depth 1: [0,0.45], n=(45,12)
**PRUNE** @depth 1: [0,0.45]
r=1000 d=0.0230657 1.17737; mh=3 n=(57,23)
r=2000 d=0.0171015 1.02635; mh=3 n=(57,23)

Sampling @ nn=361 pred locs:
r=1000 d=0.0194236 1.04473; mh=3 n=(60,20)
**GROW** @depth 1: [0,0.45], n=(45,12)
r=2000 d=0.0212965 0.0533747 0(0.998999); mh=3 n=(50,10,20)
r=3000 d=0.0192226 0(0.0149189) 0(1.15603); mh=3 n=(50,12,18)
r=4000 d=0.0234385 0.0988804 1.21228; mh=3 n=(48,14,18)
r=5000 d=0.0216443 0(1.4039) 0.133204; mh=3 n=(50,12,18)
Grow: 0.7072%, Prune: 0.1488%, Change: 31.02%, Swap: 46.31%
finished repetition 2 of 2
removed 3 leaves from the tree
```

Progress indicators show where the LLM (`corr=0(`$d$`)`) or the GP is active. Figure 10 shows how similar the resulting posterior predictive surfaces are compared to the treed GP (without LLM).

Finally, viewing 1-d projections of `tgp`-class output is possible by supplying a 1–vector `proj` argument to the `plot.tgp`. Figure 11 shows the two projections for `exp.btgpllm`. In the *left* surface plots the open circles indicate the mean of posterior predictive distribution. Red lines show the 90% intervals, the norm of which are shown on the *right*.

## 3.4 Motorcycle Accident Data

The Motorcycle Accident Dataset [24] is a classic nonstationary data set used in recent literature [21] to demonstrate the success of nonstationary models. The data consists of measurements of the 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 (heteroskedasticity) which makes it useful for illustrating how the treed GP model handles this nu-

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

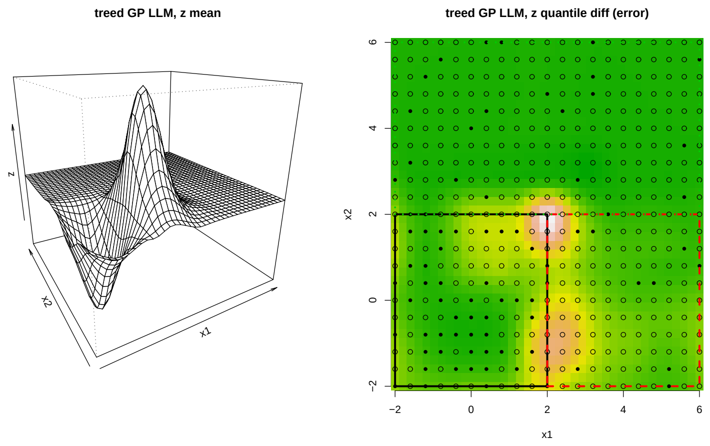


Figure 10: *Left:* posterior predictive mean using `btgpllm` on synthetic exponential data; *right* image plot of posterior predictive variance with data locations `X` (dots) and predictive locations `XX` (circles).

ance. There are at least two—perhaps three—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, verb = 0)
```

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*` functions automates this procedure. Progress indicators are suppressed.

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 the response.

```
> moto.btlm <- btlm(X = mcycle[, 1], Z = mcycle[, 2],
+     m0r1 = TRUE, verb = 0)
```

Figure 13 shows the resulting piecewise linear predictive surface and MAP partition ($\hat{\mathcal{T}}$).

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

```
> plot(exp.btgpllm, main = "treed GP LLM,", proj = c(1))
> plot(exp.btgpllm, main = "treed GP LLM,", proj = c(2))
```

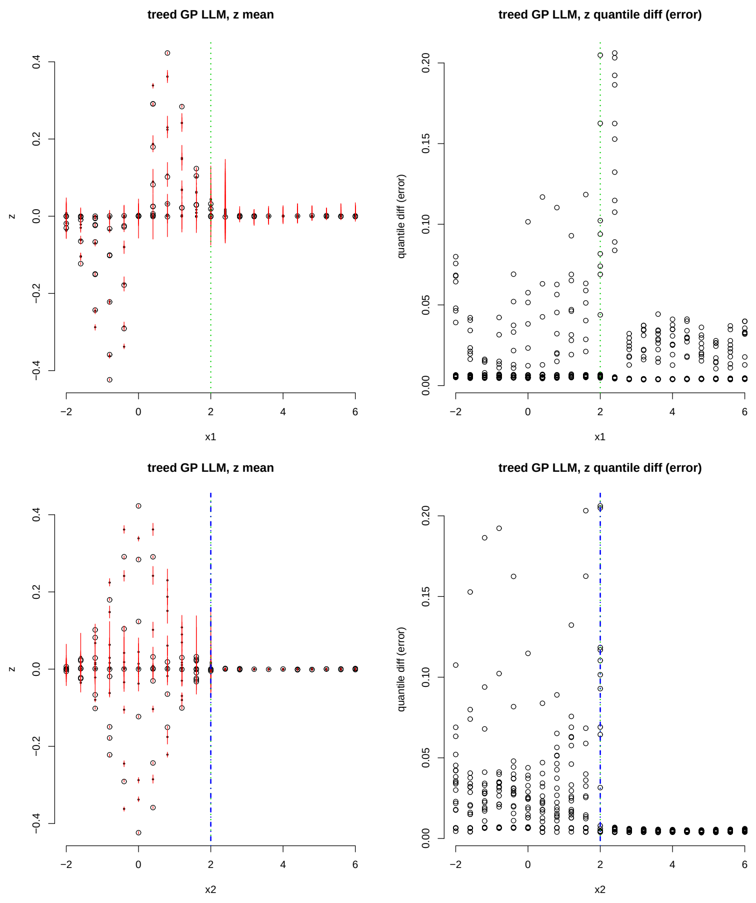


Figure 11: 1-d projections of the posterior predictive surface (*left*) and normed predictive intervals (*right*) of the 1-d tree GP LLM analysis of the synthetic exponential data. The *top* plots show projection onto the first input, and the *bottom* ones show the second.

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*` functions can force an i.i.d. hierarchical linear model by setting `bprior="b0"`.

```
> moto.btgpllm <- btgpllm(X = mcycle[, 1], Z = mcycle[,
+     2], bprior = "b0", m0r1 = TRUE, verb = 0)
> moto.btgpllm.p <- predict(moto.btgpllm)
```

The `predict.tgp` function obtains posterior predictive estimates from the MAP

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

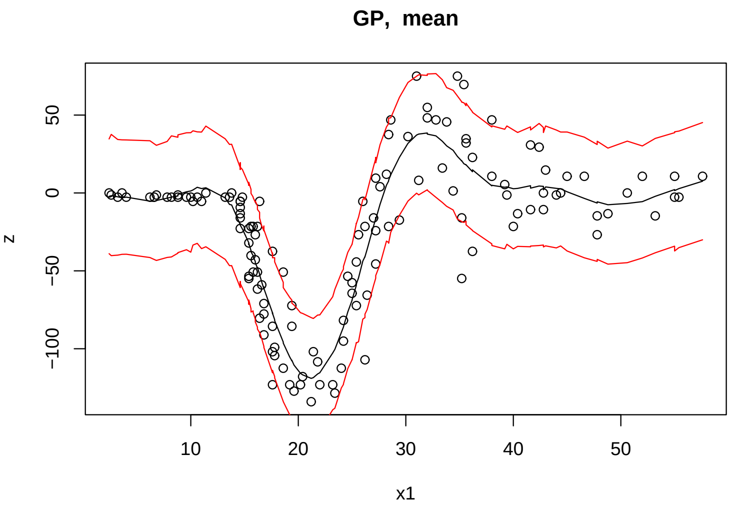


Figure 12: Posterior predictive distribution using `bgp` on the motorcycle accident data: mean and 90% credible interval

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

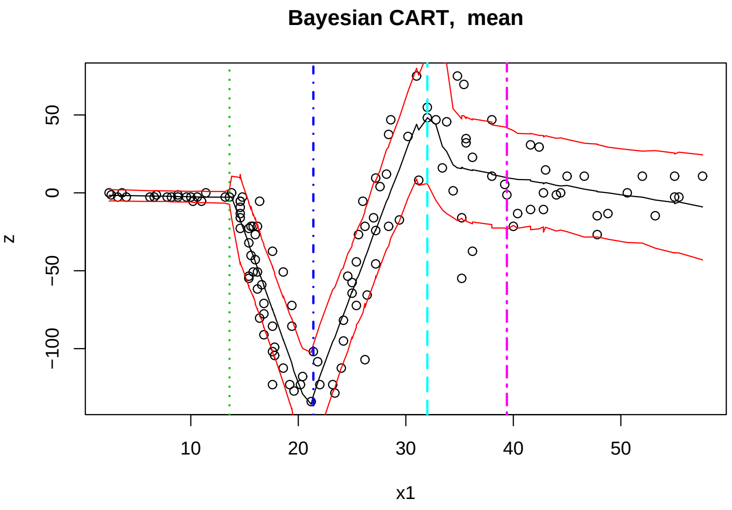
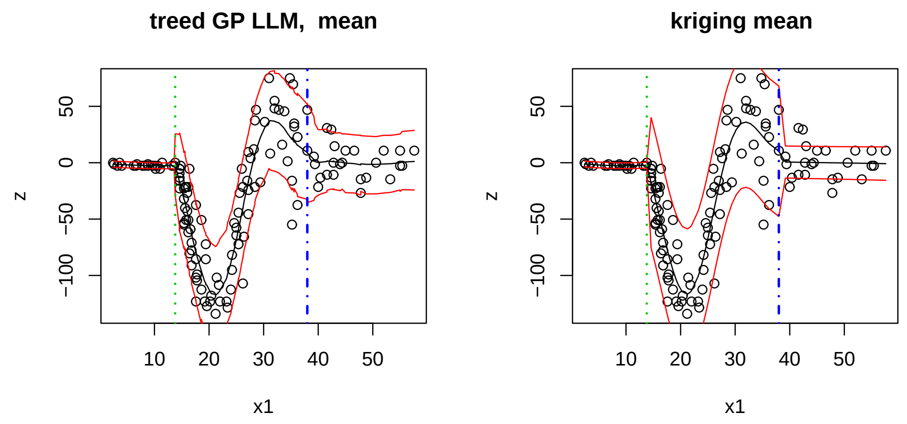


Figure 13: Posterior predictive distribution using `btlm` on the motorcycle accident data: mean and 90% credible interval

parameterization (a.k.a., *kriging*). The resulting posterior predictive surface is shown in the *top–left* of Figure 14. The *bottom–left* 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. The *right*–hand side of the figure shows the MAP surfaces obtained from the output of the

```
> par(mfrow = c(1, 2))
> plot(moto.btgpllm, main = "treed GP LLM,", layout = "surf")
> plot(moto.btgpllm.p, center = "km", layout = "surf")
```



```
> par(mfrow = c(1, 2))
> plot(moto.btgpllm, main = "treed GP LLM,", layout = "as")
> plot(moto.btgpllm.p, as = "ks2", layout = "as")
```

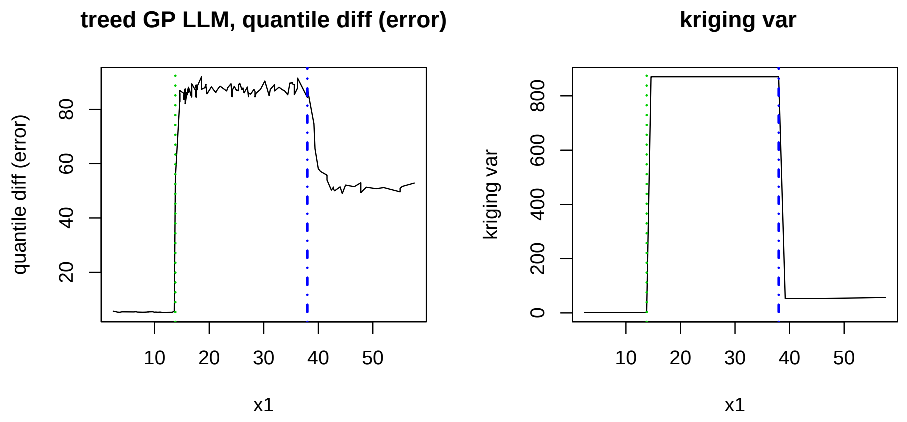


Figure 14: *top* Posterior predictive distribution using treed GP LLM on the motorcycle accident data. The *left*–hand panes how mean and 90% credible interval; *bottom* Quantile-norm (90%-5%) showing input-dependent noise. The *right*–hand panes show similar *kriging* surfaces for the MAP parameterization.

`predict.tgp` function.

The `tgp`–default `bprior="bflat"` implies an improper prior on the regression coefficients $\boldsymbol{\beta}$. It essentially forces $\mathbf{W} = \infty$, thus eliminating the need to specify priors on $\boldsymbol{\beta}_0$ of $\mathbf{W}^{-1}$ in (1). This was chosen as the default because it works well in many examples, and leads to a simpler overall model and a faster implementation. However, the Motorcycle data is an exception. Moreover, when the response data is very noisy (i.e., low signal–to–noise ratio), `tgp` can be expected to partition heavily under the `bprior="bflat"` prior. In such cases, one of the other proper priors like the full hierarchical `bprior="b0"` or the independent `bprior="b0tau"` might be preferred.

## 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) [10]. 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

$$E(Z|\mathbf{x}) = \mu = 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5, \tag{18}$$

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. [5] used this data to compare their treed LM 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)

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

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]
```

This example compares Bayesian treed LMs 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 treed LM, and obtain the RMSE.

```
> fr.btlm <- btlm(X = X, Z = Z, XX = XX, tree = c(0.95,
+     2), pred.n = FALSE, m0r1 = TRUE, verb = 0)
> fr.btlm.mse <- sqrt(mean((fr.btlm$ZZ.mean - ff$Ytrue)^2))
> fr.btlm.mse
```

```
[1] 1.939446
```

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

```
> fr.bgpllm <- bgpllm(X = X, Z = Z, XX = XX, pred.n = FALSE,
+     m0r1 = TRUE, verb = 0)
> fr.bgpllm.mse <- sqrt(mean((fr.bgpllm$ZZ.mean - ff$Ytrue)^2))
> fr.bgpllm.mse

[1] 0.4241515
```

So, the GP LLM is 4.573 times better than Bayesian treed LM on this data, in terms of RMSE (in terms of MSE the GP LLM is 2.138 times better).

Parameter traces need to be gathered in order to judge the ability of the GP LLM model to identify linear and irrelevant effects.

```
> XX1 <- matrix(rep(0, 10), nrow = 1)
> fr.bgpllm.tr <- bgpllm(X = X, Z = Z, XX = XX1, pred.n = FALSE,
+     trace = TRUE, verb = 0)
```

Notice that the `m0r1=TRUE` has been omitted so that the $\boldsymbol{\beta}$ estimates provided below will be on the original scale. A summary of the parameter traces show that the Markov chain had the following (average) configuration for the booleans.

```
> apply(fr.bgpllm.tr$trace$XX[[1]][, 27:36], 2, mean)

 b1  b2  b3  b4  b5  b6  b7  b8  b9 b10
  1   1   1   0   0   0   0   0   0   0
```

Therefore the GP LLM model correctly identified that only the first three input variables interact only linearly with the response. This agrees with dimension–wise estimate of the total area of the input domain under the LLM (out of a total of 10 input variables).

```
> mean(fr.bgpllm.tr$trace$linarea$ba)

[1] 7
```

A similar summary of the parameter traces for $\boldsymbol{\beta}$ shows that the GP LLM correctly identified the linear regression coefficients associated with the fourth and fifth input covariates (from (18))

```
> summary(fr.bgpllm.tr$trace$XX[[1]][, 9:10])

     beta4            beta5      
 Min.   : 8.623   Min.   :4.309  
 1st Qu.: 9.370   1st Qu.:5.176  
 Median : 9.564   Median :5.376  
 Mean   : 9.550   Mean   :5.375  
 3rd Qu.: 9.735   3rd Qu.:5.582  
 Max.   :10.431   Max.   :6.313  
```

and that the rest are much closer to zero.

```
> apply(fr.bgpllm.tr$trace$XX[[1]][, 11:15], 2, mean)

      beta6       beta7       beta8       beta9      beta10 
-0.23968561  0.37046946  0.13081722 -0.07842566  0.11911203 
```

## 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(lh = 0, dopt = 10)
> X <- exp2d.data$X
> Z <- exp2d.data$Z
> Xcand <- lhs(1000, rbind(c(-2, 6), c(-2, 6)))
```

In contrast with the data from Section 3.3, which was based on a grid, the above code generates a randomly subsampled $D$–optimal design $\mathbf{X}$ from LH candidates, and random responses $\mathbf{Z}$. As before, design configurations are more densely packed in the interesting region. Candidates $\tilde{\mathbf{X}}$ are from a large LH–sample.

Given some data $\{\mathbf{X}, \mathbf{Z}\}$, the first step in sequential design using `tgp` is to fit 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",
+     verb = 0)
```

The trees are shown in Figure 15. Then, use the `tgp.design` function to create $D$–optimal candidate designs in each region of $\hat{\mathcal{T}}$. For the purposes of illustrating the `improv` statistic, I have manually added the known (from calculus) global minimum to `XX`.

```
> XX <- tgp.design(200, Xcand, exp1)

sequential treed D-Optimal design in 3 partitions
dopt.gp (1) choosing 55 new inputs from 272 candidates
dopt.gp (2) choosing 53 new inputs from 263 candidates
dopt.gp (3) choosing 93 new inputs from 465 candidates

> XX <- rbind(XX, c(-sqrt(1/2), 0))
```

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 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 [3].

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. The following code gathers all three adaptive sampling statistics: ALM, ALC, & EI.

```
> exp.as <- btgpllm(X = X, Z = Z, XX = XX, corr = "exp",
+     improv = TRUE, Ds2x = TRUE, verb = 0)
```

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

```
NOTICE: skipped plotting tree of height 1, with lpost = 120.941
```

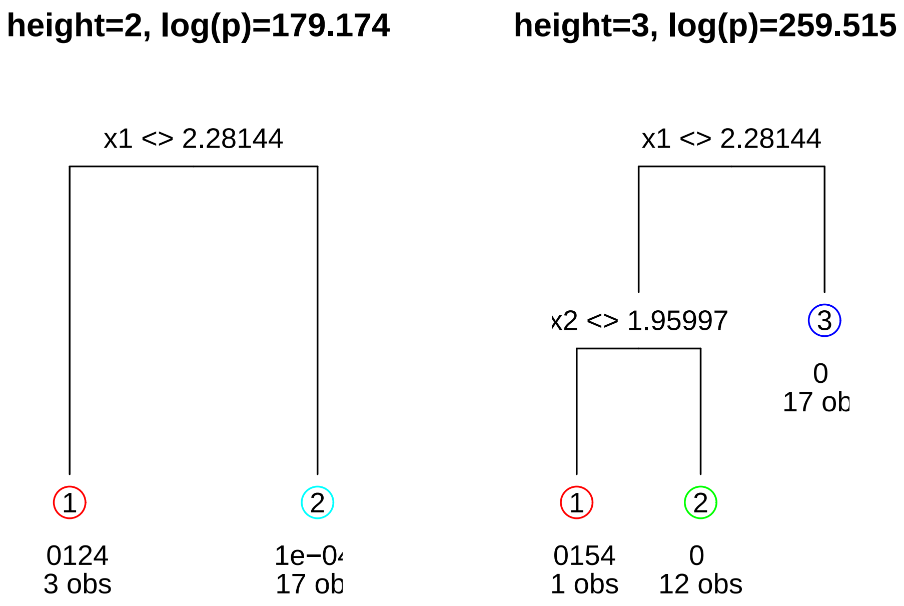


Figure 15: MAP trees of each height encountered in the Markov chain for the exponential data, showing $\hat{\sigma}^2$ and the number of observations $n$ at the leaves. $\hat{\mathcal{T}}$ is the one with the maximum $\log(p)$ above.

Figure 17 shows the posterior predictive estimates of the adaptive sampling statistics. The error surface, on the *left*, 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, these results depend heavily on the clumping of the original design in the uninteresting areas, and on the estimate of $\hat{\mathcal{T}}$. Adaptive sampling via the ALC, or EI (or both) algorithms proceeds similarly, following the surfaces shown in *center* and *right* panels of Figure 17.

## A Implementation notes

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

```
> plot(exp1$X, pch = 19, cex = 0.5)
> points(XX)
> mapT(exp1, add = TRUE)
```

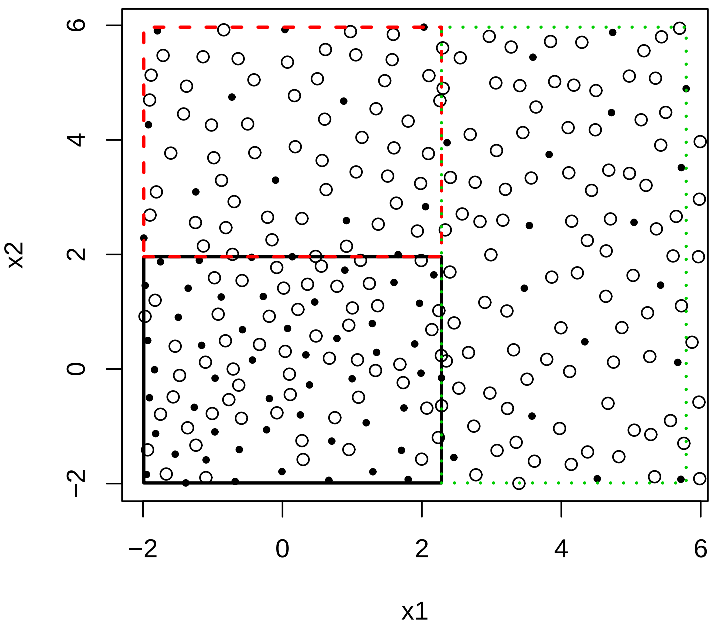


Figure 16: Treed $D$–optimal candidate locations XX (circles), input locations X (dots), and MAP tree $\hat{\mathcal{T}}$

```
> par(mfrow = c(1, 3), bty = "n")
> plot(exp.as, main = "tgpllm,", layout = "as", as = "alm")
> plot(exp.as, main = "tgpllm,", layout = "as", as = "alc")
> plot(exp.as, main = "tgpllm,", layout = "as", as = "improv")
```

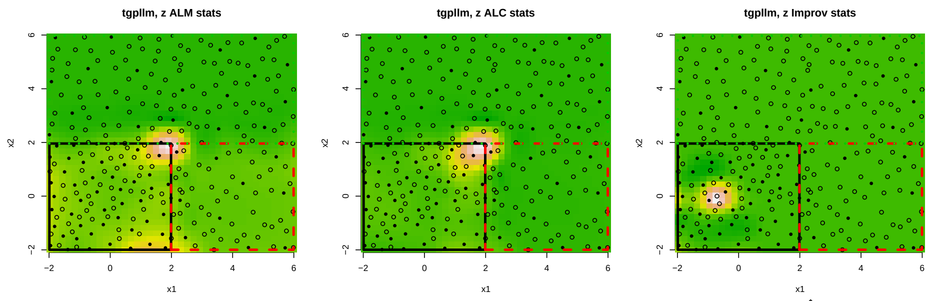


Figure 17: *Left*: Image plots of adaptive sampling statistics and MAP trees $\hat{\mathcal{T}}$; *Left*; ALM adaptive sampling image for (only) candidate locations XX (circles); *center*: ALC; and *right:* EI.

an $\Re^{m_X}$ dimensional unit cube. This makes it easier to construct prior distributions for the width parameters to the correlation function $K(\cdot,\cdot)$. 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 $g_\nu$ to the correlation function $K(\cdot,\cdot)$ in region $r_\nu$ would go as

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

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

## B Interfaces and features

The following subsections describe some of the ancillary features of the `tgp` package such as the gathering and summarizing of MCMC parameter traces, the progress meter, and an example of how to use the `predict.tgp` function in a collaborative setting.

### B.1 Parameter traces

Traces of (almost) all parameters to the `tgp` model can be collected by supplying `trace=TRUE` to the `b*` functions. In the current version traces for the linear prior correlation matrix ($\mathbf{W}$) are not provided. I shall illustrate the gathering and analyzing of traces through example. But first, a few notes and cautions.

Models which involve treed partitioning may have more than one base model (GP or LM). The process governing a particular input $\mathbf{x}$ depends on the coordinates of $\mathbf{x}$. As such, `tgp` records region–specific traces of parameters to GP (and linear) models at the locations enumerated in the `XX` argument. Even traces of single–parameter Markov chains can require hefty amounts of storage, so recording traces at each of the `XX` locations can be an enormous memory hog. A related warning will be given if the product of |`XX`|, `(BTE[2]-BTE[1])/BTE[3]` and `R` is beyond a threshold. The easiest way to keep the storage requirements for traces down is to control the size of `XX` and the thinning level `BTE[3]`. Finally, traces for most of the parameters are stored in output files. The contents of the trace files are read into `R` and stored as `data.frame` objects, and the files are removed. The existence of partially written trace files in the current working directory (CWD)—while the `C` code is executing—means that not more than one `tgp` run (with `trace = TRUE`) should be active in the CWD at one time.

Consider again the exponential data. For illustrative purposes I chose `XX` locations (where traces are gathered) to be (1) in the interior of the interesting region, (2) on/near the plausible intersection of partition boundaries, and (3) in the interior of the flat region. The hierarchical prior `bprior = "b0"` is used to leverage a (prior) belief the most of the input domain is uninteresting.

```
> exp2d.data <- exp2d.rand(n2 = 150, lh = 0, dopt = 10)
> X <- exp2d.data$X
> Z <- exp2d.data$Z
> XX <- rbind(c(0, 0), c(2, 2), c(4, 4))
```

We now fit a treed GP LLM and gather traces, and also gather EI and ALC statistics for the purposes of illustration. Prediction at the input locations `X` is turned off to be thrifty.

```
> out <- btgpllm(X = X, Z = Z, XX = XX, corr = "exp",
+     bprior = "b0", pred.n = FALSE, Ds2x = TRUE, R = 10,
+     trace = TRUE, verb = 0)
```

Figure 18 shows a dump of `out$trace` which is a `"tgptraces"`–class object. It depicts the full set of parameter traces broken down into the elements of a `list`: `$XX` with GP/LLM parameter traces for each `XX` location (the parameters are listed); `$hier` with traces for (non–input–dependent) hierarchical parameters (listed); `$linarea` recording proportions of the input space under the LLM; `$parts` with the boundaries of all partitions visited; `$post` containing (log) posterior probabilities; `preds` containing traces of samples from the posterior predictive distribution and adaptive sampling statistics.

Plots of traces are useful for assessing the mixing of the Markov chain. For example, Figure 19 plots traces of the range parameter ($d$) for each of the 3 predictive locations `XX`. It is easy to see which of the locations is in the same partition with others, and which have smaller range parameters than others.

The mean area under the LLM can be calculated as

```
> linarea <- mean(out$trace$linarea$la)
> linarea

[1] 0.530641
```

This means that the expected proportion of the input domain under the full LLM is 0.531. Figure 20 shows a histogram of areas under the LLM. The clumps near 0, 0.25, 0.5, and 0.75 can be thought of as representing quadrants (none, one, two, and tree) under the LLM. Similarly, we can calculate the probability that each of the `XX` locations is governed by the LLM.

```
> m <- matrix(0, nrow = length(trXX), ncol = 3)
> for (i in 1:length(trXX)) m[i, ] <- as.double(c(out$XX[i,
+     ], mean(trXX[[i]]$b)))
> m <- data.frame(cbind(m, 1 - m[, 3]))
> names(m) = c("XX1", "XX2", "b", "pllm")
> m

  XX1 XX2       b    pllm
1   0   0 1.00000 0.00000
2   2   2 0.64852 0.35148
3   4   4 0.50384 0.49616
```

```
> out$trace

This 'tgptraces'-class object contains traces of the parameters
to a tgp model. Access is as a list:

1.) $XX contains the traces of GP parameters for 3 predictive
    locations

    Each of $XX[[1]] ... $XX[[3]]  is a data frame with the
    columns representing GP parameters:

 [1] index  lambda s2     tau2   beta0  beta1  beta2  nug
 [9] d      b      ldetK

2.) $hier has a trace of the hierarchical params:

 [1] s2.a0   s2.g0   tau2.a0 tau2.g0 beta0   beta1   beta2
 [8] d.a0    d.g0    d.a1    d.g1    nug.a0  nug.g0  nug.a1
[15] nug.g1

3.) $linarea has a trace of areas under the LLM.  It is a
    data frame with columns:

    count: number of booleans b=0, indicating LLM
       la: area of domain under LLM
       ba: area of domain under LLM weighed by dim

4.) $parts contains all of the partitions visited.  Use the
    tgp.plot.parts.[1d,2d] functions for visuals

5.) $post is a data frame with columns showing the following:
    log posterior ($lpost), tree height ($height), IS
    weights ($w), tempered log posterior ($tlpost), inv-temp
    ($itemp), and weights adjusted for ESS ($wess)

6.) $preds is a list containing data.frames for samples from
    the posterior predictive distributions data (X) locations
    (if pred.n=TRUE: $Zp, $Zp.km, $Zp.ks2) and (XX) locations
    (if XX != NULL: $ZZ, $ZZ.km, $ZZ.ks2), with $Ds2x when
    input argument ds2x=TRUE, and $improv when improv=TRUE
```

Figure 18: Listing the contents of `"tgptraces"`–class objects.

```
> trXX <- out$trace$XX
> ltrXX <- length(trXX)
> y <- trXX[[1]]$d
> for (i in 2:ltrXX) y <- c(y, trXX[[i]]$d)
> plot(log(trXX[[1]]$d), type = "l", ylim = range(log(y)),
+     ylab = "log(d)", main = "range (d) parameter traces")
> names <- "XX[1,]"
> for (i in 2:ltrXX) {
+     lines(log(trXX[[i]]$d), col = i, lty = i)
+     names <- c(names, paste("XX[", i, ",]", sep = ""))
+ }
> legend("bottomleft", names, col = 1:ltrXX, lty = 1:ltrXX)
```

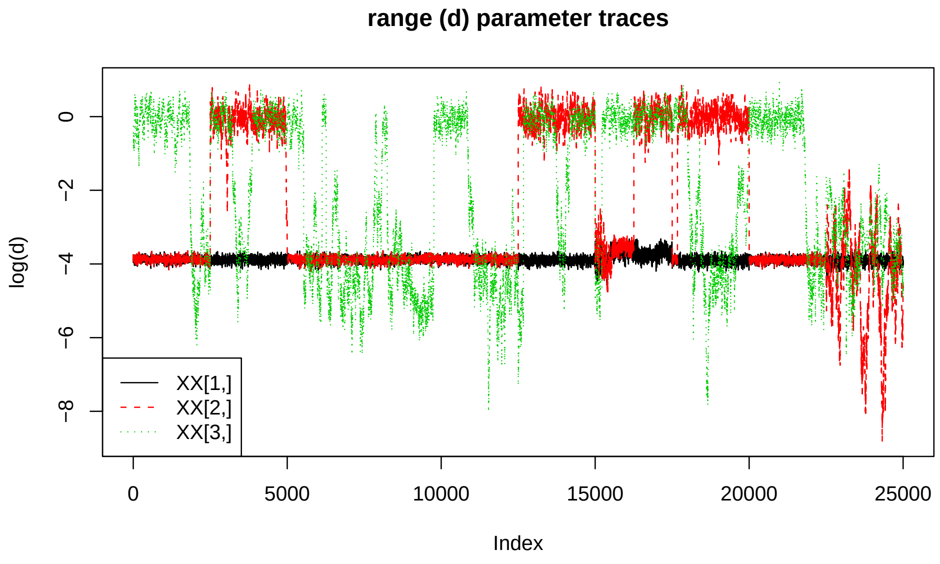


Figure 19: Traces of the (log of the) first range parameter for each of the three XX locations

The final column above represents the probability that the corresponding XX location is under the LLM (which is equal to 1-b).

Traces of posterior predictive and adaptive sampling statistics are contained in the $preds field. For example, Figure 21 shows samples of the ALC statistic $\Delta\sigma^2(\tilde{\mathbf{x}})$. We can see from the trace that statistic is generally lowest for XX[3,] which is in the uninteresting region, and that there is some competition between XX[2,] which lies on the boundary between the regions, and XX[1,] which is in the interior of the interesting region. Similar plots can be made for the other adaptive sampling statistics (i.e., ALM & EI).

```
> hist(out$trace$linarea$la)
```

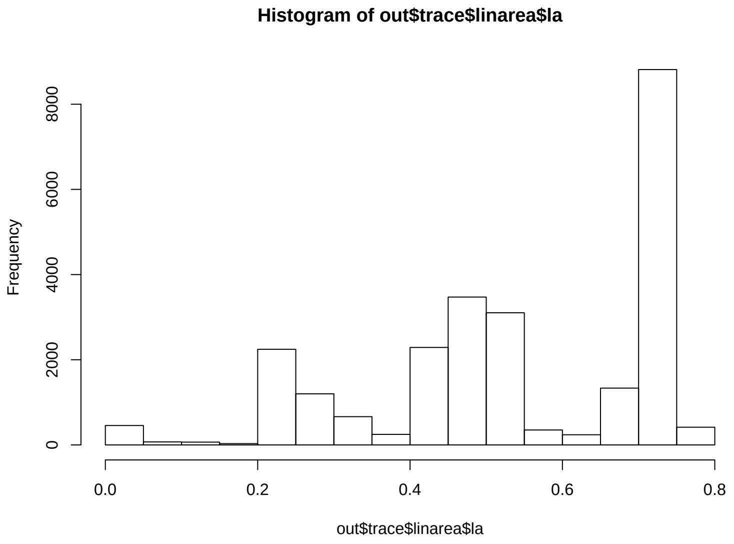


Figure 20: Histogram of proportions of the area of the input domain under the LLM

## B.2 Explaining the progress meter

The progress meter shows the progress of the MCMC as it iterates through the desired number of rounds of burn–in (`BTE[1]`), and sampling (`BTE[2]-BTE[1]`), for the requested number of repeats (`R`). The verbosity of progress meter print statements is controlled by the `verb` arguments to the `b*` functions. Providing `verb=0` silences all non–warning (or error) statements. To suppress warnings, try enclosing commands within `suppressWarnings(...)`, or globally set `options(warn=0)`. See the help file (`?options`) for more global warning settings.

The default verbosity setting (`verb=1`) shows all *grows* and *prunes*, and a summary of $d$–(range) parameters for each partition every 1000 rounds. Higher verbosity arguments will show more tree operations, e.g., *change* and *swap*, etc. Setting `verb=2` will cause an echo of the `tgp` model parameters and their starting values; but is otherwise the same as `verb=1`. The max is `verb=4` shows all successful tree operations. Here is an example *grow* statement.

```
**GROW** @depth 2: [0,0.05], n=(10,29)
```

The `*GROW*` statements indicate the depth of the split leaf node; the splitting dimension $u$ and location $v$ is shown between square brackets `[u,v]`, followed by the size of the two new children `n=(n1,n2)`. `*PRUNE*` is about the same, without printing `n=(n1,n2)`.

Every 1000-rounds a progress indicator is printed. Its format depends on a number of things: (1) whether parallelization is turned on or not, (2) the

```
> trALC <- out$trace$preds$Ds2x
> y <- trALC[, 1]
> for (i in 2:ncol(trALC)) y <- c(y, trALC[, i])
> plot(log(trALC[, 1]), type = "l", ylim = range(log(y)),
+     ylab = "Ds2x", main = "ALC: samples from Ds2x")
> names <- "XX[1,]"
> for (i in 2:ncol(trALC)) {
+     lines(log(trALC[, i]), col = i, lty = i)
+     names <- c(names, paste("XX[", i, ",]", sep = ""))
+ }
> legend("bottomright", names, col = 1:ltrXX, lty = 1:ltrXX)
```

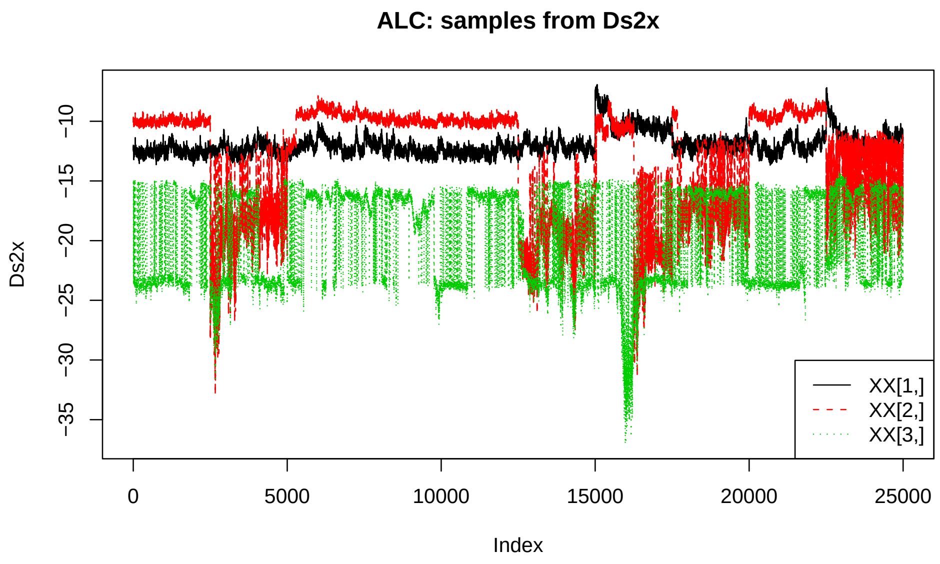


Figure 21: Traces of the (log of the) samples for the ALC statistic $\Delta\sigma^2(\tilde{\mathbf{x}})$ at for each of the three `XX` locations

correlation model [isotropic or separable], (3) whether jumps to the LLM are allowed. Here is an example with the 2-d exp data with parallel prediction under the separable correlation function:

```
(r,l)=(5000,104) d=[0.0144 0.0236] [1.047 0/0.626]; mh=2 n=(59,21)
```

The first part `(r,l)=(5000,104)` is indicating the MCMC round number r=5000 and the number of leaves waiting to be ”consumed” for prediction by the parallel prediction thread. When parallelization is turned off (default), the print will simply be `"r=5000"`.

The second part is a printing of the $d$–(range) parameter to a separable correlation function. For 2 partitions there are two sets of square brackets. Inside the square brackets is the $m_X$ (2 in this case) range parameters for the separable correlation function. Whenever the LLM governs one of the input dimensions a zero will appear. I.e., the placement of `0/0.626` indicates the LLM is active in the 2nd dimension of the 2nd partition. 0.626 is the *d*–(range) parameter that would have been used if the LLM were inactive. Whenever all dimensions are under the LLM, the d-parameter print is simply `[0]`. This also happens when forcing the LLM (i.e., for `blm` and `btlm`), where `[0]` appears for each partition. These prints will look slightly different if the isotropic instead of separable correlation is used, since there are not as many range parameters.

### B.3 Collaboration with `predict.tgp`

In this section I revisit the motorcycle accident data in order to demonstrate how the `predict.tgp` function can be helpful in collaborative uses of `tgp`. Consider a fit of the motorcycle data, and suppose that we do inference for the model parameters only (obtaining no samples from the posterior predictive distribution. The `"tgp"`-class output object can be saved to a file using the `R`–internal `save` function.

```
> library(MASS)
> out <- btgpllm(X = mcycle[, 1], Z = mcycle[, 2],
+     bprior = "b0", m0r1 = TRUE, pred.n = FALSE, verb = 0)
> save(out, file = "out.Rsave")
> out <- NULL
```

Note that there is nothing to plot here because there is no predictive data. (`out <- NULL` is set for illustrative purposes.)

Now imagine e–mailing the "out.Rsave" file to a collaborator who wishes to use your fitted `tgp` model. S/he could first load in the `"tgp"`–class object we just saved, design a new set of predictive locations `XX` and obtain kriging estimates from the MAP model.

```
> load("out.Rsave")
> XX <- seq(2.4, 56.7, length = 200)
> out.kp <- predict(out, XX = XX, pred.n = FALSE)
```

Another option would be to sample from the posterior predictive distribution of the MAP model.

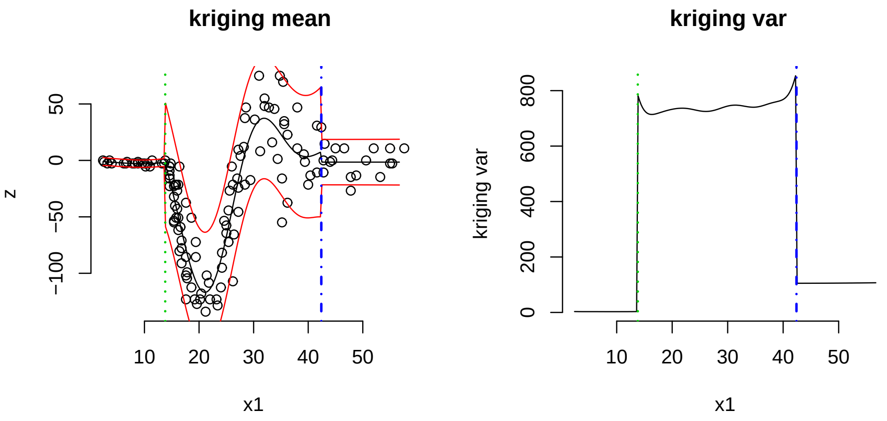
```
> out.p <- predict(out, XX = XX, pred.n = FALSE, BTE = c(0,
+     1000, 1))
```

This holds the parameterization of the `tgp` model *fixed* at the MAP, and samples from the GP or LM posterior predictive distributions at the leaves of the tree.

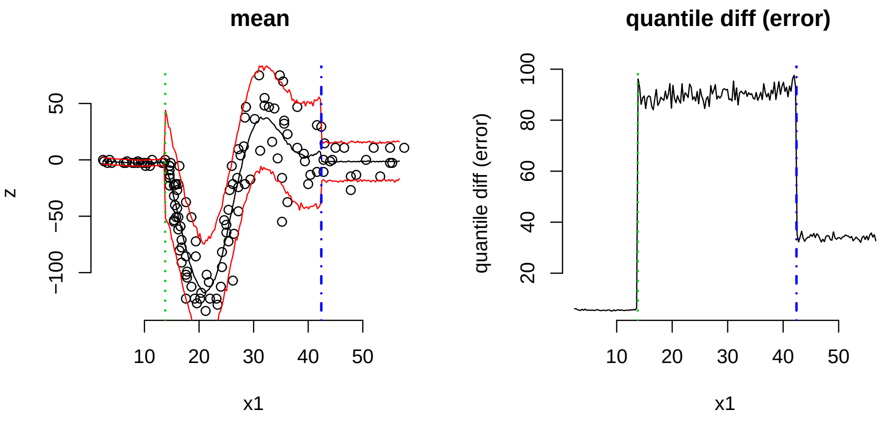
Finally, the MAP parameterization can be used as a jumping-off point for more sampling from the joint posterior and posterior predictive distribution.

```
> out2 <- predict(out, XX, pred.n = FALSE, BTE = c(0,
+     2000, 2), krige = FALSE)
```

```
> plot(out.kp, center = "km", as = "ks2")
```



```
> plot(out.p)
```



```
> plot(out2)
```

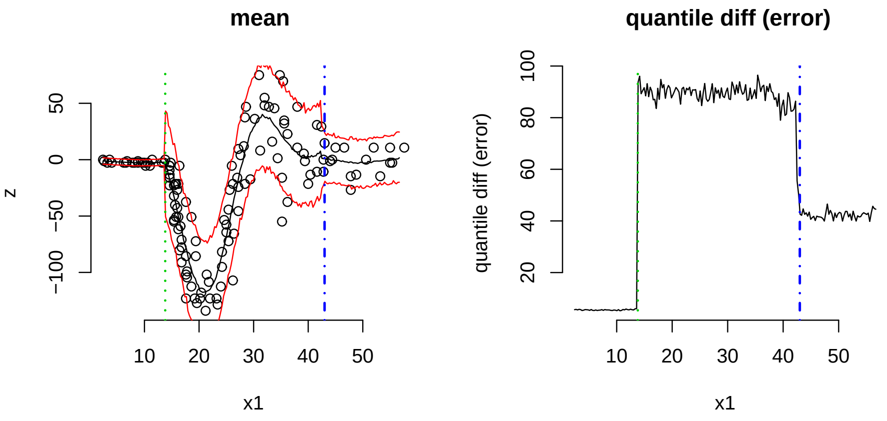


Figure 22: Predictive surfaces (*left*) and error/variance plots (*right*) resulting from three different uses of the `predict.tgp` function: MAP kriging (*top*), sampling from the MAP (*middle*), sampling from the joint posterior and posterior predictive starting from the MAP (*bottom*).

Since the return–value of a `predict.tgp` call is also a `"tgp"`–class object the process can be applied iteratively. That is, `out2` can also be passed to `predict.tgp`.

Figure 22 plots the posterior predictive surfaces for each of the three calls to `predict.tgp` above. The kriging surfaces are smooth within regions of the partition, but the process is discontinuous across partition boundaries. The middle surface is simply a Monte Carlo–sample summarization of the kriging one above it. The final surface summarizes samples from the posterior predictive distribution when obtained jointly with $\mathcal{T}|\boldsymbol{\theta}$ and $\boldsymbol{\theta}|\mathcal{T}$. Though these summaries are still "noisy" they depict a process with smoother transitions across partition boundaries than ones conditioned only on the MAP.

The `predict.tgp` function can also sample from the ALC statistic and calculate expected improvements (EI) at the `XX` locations.

## C Configuration and performance optimization

In what follows I describe customizations and enhancements that can be made to `tgp` at compile time in order to take advantage of custom computing architectures. The compilation of `tgp` with a linear algebra library different from the one used to compile `R` (e.g., ATLAS), and the configuration and compilation of `tgp` with parallelization is described in detail.

### C.1 Linking to ATLAS

`ATLAS` [26] is supported as an alternative to standard BLAS and LAPACK for fast, automatically tuned, linear algebra routines. Compared to standard `BLAS/Lapack`, those automatically tuned by `ATLAS` are significantly faster. If you know that `R` has already been linked to tuned linear algebra libraries (e.g., on `OSX`), then compiling with `ATLAS` as described below, is unnecessary—just install `tgp` as usual. As an alternative to linking `tgp` to `ATLAS` directly, one could re-compile all of `R` linking it to `ATLAS`, or some other platform–specific `BLAS/Lapack`, i.e., `Intel`'s Math Kernel Library, or `AMD`'s Core Math Library, as described in:

`http://cran.r-project.org/doc/manuals/R-admin.html`

Look for the section titled "Linear Algebra". While this is arguably best solution since all of `R` benefits, the task can prove challenging to accomplish and may require administrator (root) privileges. Linking `tgp` with `ATLAS` directly is described here.

Three easy steps (assuming, of course, you have already compiled and installed `ATLAS` – `http://math-atlas.sourceforge.net`) need to be performed before you install the `tgp` package from source.

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

```
PKG_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 may need to add `"-lF77blas"`, if you compiled with `Pthreads`, you would might use

```
-llapack -lptcblas -lptf77blas -latlas
```

2. Continue editing src/Makevars. Add:

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

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

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

Now simply install the `tgp` package as usual. Reverse the above instructions to disable `ATLAS`. Don't forget to re-install the package when you're done.

## C.2 Parallelization with `Pthreads`

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 `PThreads` in order to take advantage of multiple processors, and get speed-ups of at least a factor of two. This is particularly relevant since dual processor workstations and multi-processor servers are becoming commonplace in modern research labs. However, multi–processors are not yet ubiquitous, so parallel–`tgp` is not yet the default. Using the parallel version will be slower than the non–parallel (serial) version on a single processor machine.

Enabling parallelization requires two simple steps, and then a re–install.

1. Add `-DPARALLEL` to `PKG_CXXFLAGS` of src/Makevars
2. You may need to add `-pthread` to `PKG_LIBS` of src/Makevars, or whatever is needed by your compiler in order to correctly link code with `PThreads`.

The biggest improvement in the parallel version, over the serial, is observed when calculating ALC statistics, which require $O(n_2^2)$ time for $n_2$ predictive locations, or when calculating ALM (default) or EI statistics on predictive locations whose number ($n_2$) is at least an order of magnitude larger ($n_2 \gg n_1$) than the number of input locations ($n_1$).

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 of the `tgp` package.

## References

[1] Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables.* Dover, New York, 9th dover printing, 10th gpo printing– edition, 1964.

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

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

[4] 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.

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

[6] 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.

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

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

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

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

[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. Adaptive design of supercomputer experiments. Technical report, Dept. of Applied Math & Statistics, University of California, Santa Cruz, 2006.

[13] Robert B. Gramacy and Herbert K H. Lee. Bayesian treed Gaussian process models. Technical report, Dept. of Applied Math & Statistics, University of California, Santa Cruz, 2006.

[14] 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.

[15] D.R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black box functions. *Journal of Global Optimization*, 13:455–492, 1998.

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

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

[18] 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.

[19] C.J. Paciorek. *Nonstationary Gaussian Processes for Regression and Spatial Modelling*. PhD thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania, 2003.

[20] 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.

[21] 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.

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

[23] 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.

[24] 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.

[25] Michail L. Stein. *Interpolation of Spatial Data*. Springer, New York, NY, 1999.

[26] R. Clint Whaley and Antoine Petitet. `ATLAS` (Automatically Tuned Linear Algebra Software). http://math-atlas.sourceforge.net/, 2004.