Categorical inputs, sensitivity analysis, optimization and importance tempering with `tgp` version 2, an R package for treed Gaussian process models

Robert B. Gramacy
Booth School of Business
The University of Chicago
rbgramacy@chicagobooth.edu

Matthew Taddy
Booth School of Business
The University of Chicago
taddy@chicagobooth.edu

October 13, 2012

Abstract

This document describes the new features in version 2.x of the `tgp` package for R, implementing treed Gaussian process (GP) models. The topics covered include methods for dealing with categorical inputs and excluding inputs from the tree or GP part of the model; fully Bayesian sensitivity analysis for inputs/covariates; sequential optimization of black-box functions; and a new Monte Carlo method for inference in multi-modal posterior distributions that combines simulated tempering and importance sampling. These additions extend the functionality of `tgp` across all models in the hierarchy: from Bayesian linear models, to CART, to treed Gaussian processes with jumps to the limiting linear model. It is assumed that the reader is familiar with the baseline functionality of the package, outlined in the first vignette [11].

Intended audience

The `tgp` package contains implementations of seven related Bayesian regression frameworks which combine treed partition models, linear models (LM), and stationary Gaussian process (GP) models. GPs are flexible (phenomenological) priors over functions which, when used for regression, are usually relegated to smaller applications for reasons of computational expense. Trees, by contrast, are a crude but efficient divide-and-conquer approach to non-stationary regression. When combined they are quite powerful, and provide a highly flexible nonparametric and non-stationary family of regression tools. These treed GP models have been successfully used in a variety of contexts, in particular in the sequential design and analysis of computer experiments.

The models, and the (base) features of the package, are described the vignette for version 1.x of the package [11]. This document is intended as a follow-on, describing four new features that have been added to the package in version 2.x. As such, it is divided into four essentially disjoint sections: on categorical inputs (Section 1), sensitivity analysis (Section 2), statistical optimization (Section 3), and importance tempering (Section 4). The ability to deal with categorical inputs greatly expands the sorts of regression problems which `tgp` can handle. It also enables the partition component of the model to more parsimoniously describe relationships that were previously left to the GP part of the model, at a great computational expense and interpretational disadvantage. The analysis of sensitivity to inputs via the predictive variance enables the user to inspect, and understand, the first-order and total effects of each of the inputs on the response. The section on statistical optimization expands the sequential design feature set described in the first vignette. We now provide a skeleton which automates the optimization of black-box functions by expected improvement, along with tools and suggestions for assessing convergence. Finally, the addition of tempering-based MCMC methods leads to more reliable inference via a more thorough exploration of the highly multi-modal posterior distributions that typically result from tree based models, which previously could only be addressed by random restarts. Taken all together, these four features have greatly expanded the capabilities of the package, and thus the variety of statistical problems which can be addressed with the `tgp` family of methods.

Each of the four sections to follow will begin with a short mathematical introduction to the new feature or methodology and commence with extensive examples in R on synthetic and real data. 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. As with the first vignette, the R code in each of the sections to follow is also available as a demo in the package. 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`

Each section starts by seeding the random number generator with `set.seed(0)`. This is done to make the results and analyses reproducible within this document (assuming identical architecture [64-bit Linux] and version of R [2.10.1]), and in demo form. We 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.

1 Non–real–valued, categorical and other inputs

Early versions of `tgp` worked best with real–valued inputs $\mathbf{X}$. While it was possible to specify ordinal, integer–valued, or even binary inputs, `tgp` would treat them the same as any other real–valued input. Two new arguments to `tgp.default.params`, and thus the ellipses (`...`)

argument to the `b*` functions, provide a more natural way to model with non–real valued inputs. In this section we shall introduce these extensions, and thereby illustrate how the current version of the package can more gracefully handle categorical inputs. We argue that the careful application of this new feature can lead to reductions in computational demands, improved exploration of the posterior, increased predictive accuracy, and more transparent interpretation of the effects of categorical inputs.

Classical treed methods, such as CART [1], can cope quite naturally with categorical, binary, and ordinal, inputs. Categorical inputs can be encoded in binary, and splits can be proposed with rules such as $x_i < 1$. Once a split is made on a binary input, no further process is needed, marginally, in that dimension. Ordinal inputs can also be coded in binary, and thus treated as categorical, or treated as real–valued and handled in a default way. GP regression, however, handles such non–real–valued inputs less naturally, unless (perhaps) a custom and non–standard form of the covariance function is used [27]. When inputs are scaled to lie in $[0, 1]$, binary–valued inputs x_i are always a constant distance apart—at the largest possible distance in the range. A separable correlation function width parameter d_i will tend to infinity (in the posterior) if the output does not vary with x_i, and will tend to zero if it does. Clearly, this functionality is more parsimoniously achieved by partitioning, e.g., using a tree. However, trees with fancy regression models at the leaves pose other problems, as discussed below.

Consider as motivation, the following modification of the Friedman data [6] (see also Section 3.5 of [11]). Augment 10 real–valued covariates in the data ($\mathbf{x} = \{x_1, x_2, \ldots, x_{10}\}$) with one categorical indicator $I \in \{1, 2, 3, 4\}$ that can be encoded in binary as

$$1 \equiv (0,0,0) \qquad 2 \equiv (0,0,1) \qquad 3 \equiv (0,1,0) \qquad 4 \equiv (1,0,0).$$

Now let the function that describes the responses (Z), observed with standard Normal noise, have a mean

$$E(Z|\mathbf{x}, I) = \begin{cases} 10\sin(\pi x_1 x_2) & \text{if } I = 1 \\ 20(x_3 - 0.5)^2 & \text{if } I = 2 \\ 10x_4 + 5x_5 & \text{if } I = 3 \\ 5x_1 + 10x_2 + 20(x_3 - 0.5)^2 + 10\sin(\pi x_4 x_5) & \text{if } I = 4 \end{cases} \tag{1}$$

that depends on the indicator I. Notice that when $I = 4$ the original Friedman data is recovered, but with the first five inputs in reverse order. Irrespective of I, the response depends only on $\{x_1, \ldots, x_5\}$, thus combining nonlinear, linear, and irrelevant effects. When $I = 3$ the response is linear $\mathbf{x}$.

A new function has been included in the `tgp` package which facilitates generating random realizations from (1). Below we obtain 500 such random realizations for training purposes, and a further 1000 for testing.

```
> fb.train <- fried.bool(500)
> X <- fb.train[,1:13]; Z <- fb.train$Y
> fb.test <- fried.bool(1000)
> XX <- fb.test[,1:13]; ZZ <- fb.test$Ytrue
```

A separation into training and testing sets will be useful for later comparisons by RMSE. The names of the data frame show that the first ten columns encode $\mathbf{x}$ and columns 11–13 encode the boolean representation of I.

```
> names(X)
```

```
 [1] "X.1"  "X.2"  "X.3"  "X.4"  "X.5"  "X.6"  "X.7"  "X.8"
 [9] "X.9"  "X.10" "I.1"  "I.2"  "I.3"
```

One, naïve approach to fitting this data would be to fit a treed GP LLM model ignoring the categorical inputs. But this model can only account for the noise, giving high RMSE, and so is not illustrated here. Clearly, the indicators must be included. One simple way to do so would be to posit a Bayesian CART model.

```
> fit1 <- bcart(X=X, Z=Z, XX=XX, verb=0)
> rmse1 <- sqrt(mean((fit1$ZZ.mean - ZZ)^2))
> rmse1
```

```
[1] 2.597121
```

In this case the indicators are treated appropriately (as indicators), but in some sense so are the real–valued inputs as only constant models are fit at the leaves of the tree. Figure 1 shows

```
> tgp.trees(fit1, "map")
```

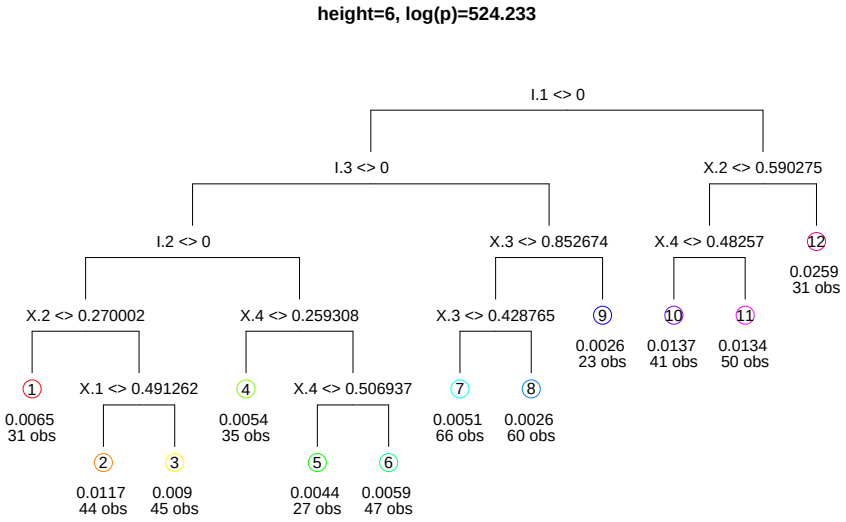

Figure 1: Diagrammatic depiction of the maximum *a' posteriori* (MAP) tree for the boolean indicator version of the Friedman data in Eq. (1) using Bayesian CART.

that the tree does indeed partition on the indicators, and the other inputs, as expected.

One might expect a much better fit from a treed linear model to this data, since the response is linear in some of its inputs.

```
> fit2 <- btlm(X=X, Z=Z, XX=XX, verb=0)
> rmse2 <- sqrt(mean((fit2$ZZ.mean - ZZ)^2))
> rmse2

[1] 2.570726
```

Unfortunately, this is not the case—the RMSE obtained is similar to the one for the CART model. Figure 2 shows that the tree does indeed partition, but not on the indicator variables.

```
> tgp.trees(fit2, "map")
```

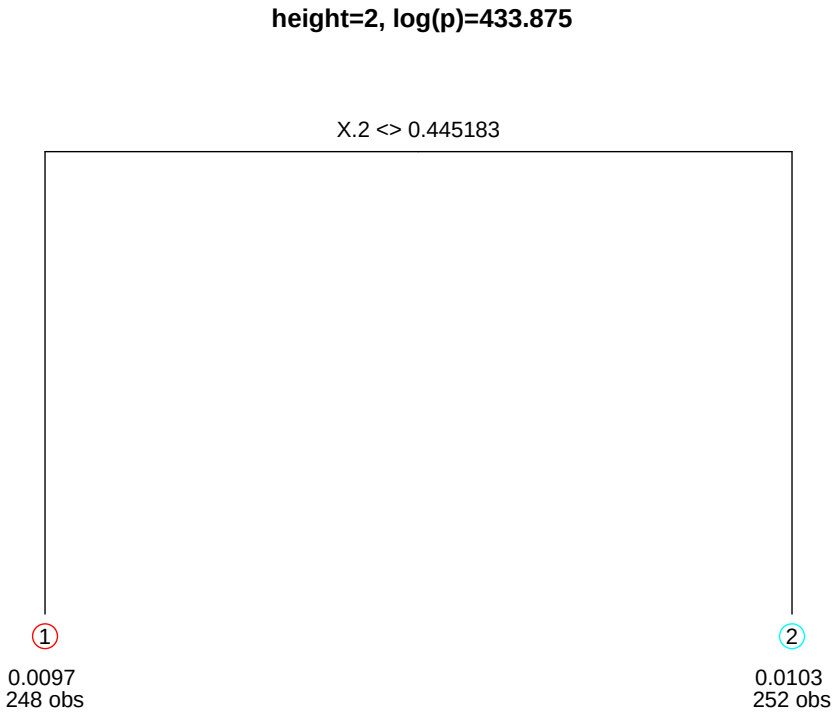

Figure 2: Diagrammatic depiction of the maximum *a' posteriori* (MAP) tree for the boolean indicator version of the Friedman data in Eq. (1) using a Bayesian treed linear model.

When a linear model is used at the leaves of the tree the boolean indicators cannot be partitioned upon because doing so would cause the design matrix to become rank–deficient at the leaves of the tree (there would be a column of all zeros or all ones). A treed GP would have the same problem.

A new feature in `tgp` makes dealing with indicators such as these more natural, by including them as candidates for treed partitioning, but ignoring them when it comes to fitting the models at the leaves of the tree. The argument `basemax` to `tgp.default.params`, and thus the ellipses (`...`) argument to the `b*` functions, allows for the specification of the last columns of `X` to be considered under the base (LM or GP) model. In the context of our example, specifying `basemax = 10` ensures that only the first 10 inputs, i.e., $\mathbf{X}$ only (excluding I), are used to predict the response under the GPs at the leaves. Both the columns

of $\mathbf{X}$ and the columns of the boolean representation of the (categorical) indicators I are (still) candidates for partitioning. This way, whenever the boolean indicators are partitioned upon, the design matrix (for the GP or LM) will not contain the corresponding column of zeros or ones, and therefore will be of full rank.

Let us revisit the treed LM model with `basemax = 10`.

```
> fit3 <- btlm(X=X, Z=Z, XX=XX, basemax=10, verb=0)
> rmse3 <- sqrt(mean((fit3$ZZ.mean - ZZ)^2))
> rmse3

[1] 1.673351


> tgp.trees(fit3, "map")
```

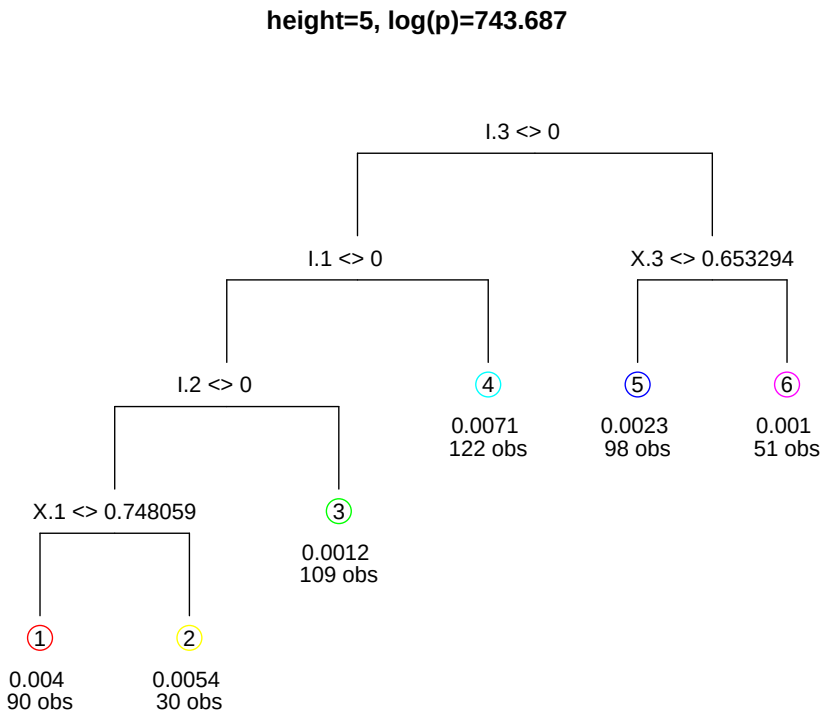

Figure 3: Diagrammatic depiction of the maximum *a' posteriori* (MAP) tree for the boolean indicator version of the Friedman data in Eq. (1) using a Bayesian treed linear model with the setting `basemax = 10`.

Figure 3 shows that the MAP tree does indeed partition on the indicators in an appropriate way—as well as on some other real–valued inputs—and the result is the lower RMSE we would expect.

A more high–powered approach would clearly be to treat all inputs as real–valued by fitting a GP at the leaves of the tree. Binary partitions are allowed on all inputs, $\mathbf{X}$ and I, but treating the boolean indicators as real–valued in the GP is clearly inappropriate since

it is known that the process does not vary smoothly over the 0 and 1 settings of the three boolean indicators representing the categorical input I.

```
> fit4 <- btgpllm(X=X, Z=Z, XX=XX, verb=0)
> rmse4 <- sqrt(mean((fit4$ZZ.mean - ZZ)^2))
> rmse4

[1] 1.26519
```

Since the design matrices would become rank–deficient if the boolean indicators are partitioned upon, there was no partitioning in this example.

```
> fit4$gpcs

  grow prune change swap
1    0    NA     NA   NA
```

Since there are large covariance matrices to invert, the MCMC inference is *very* slow. Still, the resulting fit (obtained with much patience) is better that the Bayesian CART and treed LM (with `basemax = 10`) ones, as indicated by the RMSE.

We would expect to get the best of both worlds if the setting `basemax = 10` were used when fitting the treed GP model, thus allowing partitioning on the indicators by guarding against rank deficient design matrices.

```
> fit5 <-  btgpllm(X=X, Z=Z, XX=XX, basemax=10, verb=0)
> rmse5 <- sqrt(mean((fit5$ZZ.mean - ZZ)^2))
> rmse5

[1] 1.220387
```

And indeed this is the case.

The benefits go beyond producing full rank design matrices at the leaves of the tree. Loosely speaking, removing the boolean indicators from the GP part of the treed GP gives a more parsimonious model, without sacrificing any flexibility. The tree is able to capture all of the dependence in the response as a function of the indicator input, and the GP is the appropriate non–linear model for accounting for the remaining relationship between the real–valued inputs and outputs. We can look at the maximum *a' posteriori* (MAP) tree, to see that only (and all of) the indicators were partitioned upon in Figure 4. Further advantages to this approach include speed (a partitioned model gives smaller covariance matrices to invert) and improved mixing in the Markov chain when a separable covariance function is used. Note that using a non–separable covariance function in the presence of indicators would result in a poor fit. Good range (d) settings for the indicators would not necessarily coincide with good range settings for the real–valued inputs.

A complimentary setting, `splitmin`, allows the user to specify the first column of the inputs `X` upon which treed partitioning is allowed. From Section 3.5 of the first `tgp` vignette

```
> h <- fit1$post$height[which.max(fit1$posts$lpost)]
> tgp.trees(fit5, "map")
```

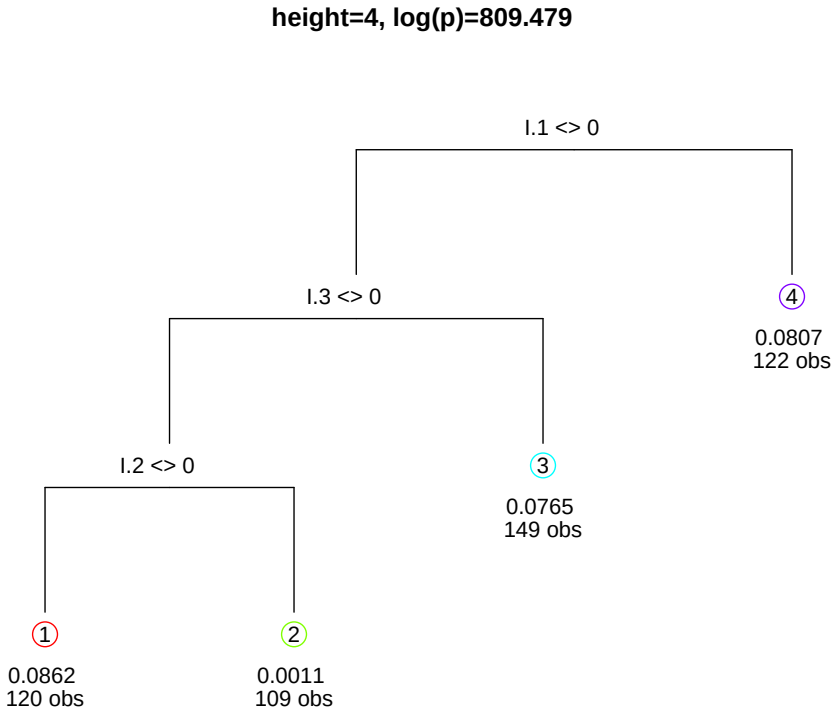

Figure 4: Diagrammatic depiction of the maximum *a' posteriori* (MAP) tree for the boolean indicator version of the Friedman data in Eq. (1) using `basemax=10`.

[11], it was concluded that the original formulation of Friedman data was stationary, and thus treed partitioning is not required to obtain a good fit. The same would be true of the response in (1) after conditioning on the indicators. Therefore, the most parsimonious model would use `splitmin = 11`, in addition to `basemax = 10`, so that only $\mathbf{X}$ are under the GP, and only I under the tree. Fewer viable candidate inputs for treed partitioning should yield improved mixing in the Markov chain, and thus lower RMSE.

```
> fit6 <-  btgpllm(X=X, Z=Z, XX=XX, basemax=10, splitmin=11, verb=0)
> rmse6 <- sqrt(mean((fit6$ZZ.mean - ZZ)^2))
> rmse6

[1] 0.3845652
```

Needless to say, it is important that the input `X` have columns which are ordered appropriately before the `basemax` and `splitmin` arguments can be properly applied. Future versions of `tgp` will have a formula–based interface to handle categorical (`factors`) and other inputs more like other R regression routines, e.g., `lm` and `glm`.

The tree and binary encodings represent a particularly thrifty way to handle categorical inputs in a GP regression framework, however it is by no means the only or best approach to

doing so. A disadvantage to the binary coding is that it causes the introduction of several new variables for each categorical input. Although they only enter the tree part of the model, and not the GP (where the introduction of many new variables could cause serious problems), this may still be prohibitive if the number of categories is large. Another approach that may be worth considering in this case involves designing a GP correlation function which can explicitly handle a mixture of qualitative (categorical) and quantitative (real-valued) factors [27]. An advantage of our treed approach is that it is straightforward to inspect the effect of the categorical inputs by, e.g., counting the number of trees (in the posterior) which contain a particular binary encoding. It is also easy to see how the categorical inputs interact with the real-valued ones by inspecting the (posterior) parameterizations of the correlation parameters in each partition on a binary encoding. Both of these are naturally facilitated by gathering traces (`trace = TRUE`), as described in the 1.x vignette [11]. In Section 2 we discuss a third way of determining the sensitivity of the response to categorical and other inputs.

2 Analysis of sensitivity to inputs

Methods for understanding how inputs, or explanatory variables, contribute to the outputs, or response, of simple statistical models are by now classic in the literature and frequently used in practical application. For example, in linear regression one can perform F–tests to ascertain the relevance of a predictor, or inspect the leverage of a particular input setting, or use Cooks' distance, to name a few. Unfortunately, such convenient statistics/methods are not available for more complicated models, such as those in the `tgp` family of nonparametric models. A more advanced tool is needed.

Sensitivity Analysis (SA) is a resolving of the sources of output variability by apportioning elements of this variation to different sets of input variables. It is applicable in wide generality. The edited volume by Saltelli et al. [29] provides an overview of the field. Valuable recent work on smoothing methods is found in [34, 4], and Storlie, et al. [35], provide a nice overview of nonparametric regression methods for inference about sensitivity. The analysis of response variability is useful in a variety of different settings. For example, when there is a large number of input variables over which an objective function is to be optimized, typically only a small subset will be influential within the confines of their uncertainty distribution. SA can be used to reduce the input space of such optimizations [36]. Other authors have used SA to assess the risk associated with dynamic factors affecting the storage of nuclear waste [14], and to investigate the uncertainty characteristics of a remote sensing model for the reflection of light by surface vegetation [21]. The `sens` function adds to `tgp` a suite of tools for global sensitivity analysis, and enables "out-of-the-box" estimation of valuable sensitivity indices for any regression relationship that may be modeled by a member of the `tgp` family.

The type of sensitivity analysis provided by `tgp` falls within the paradigm of global sensitivity analysis, wherein the variability of the response is investigated with respect to a probability distribution over the entire input space. The recent book by Saltelli et al. [30] serves as a primer on this field. Global SA is inherently a problem of statistical inference, as

evidenced by the interpolation and estimation required in a study of the full range of inputs. This is in contrast with the analytical nature of local SA, which involves derivative–based investigation of the stability of the response over a small region of inputs. We will ignore local SA for the remainder of this document.

The sensitivity of a response z to a changing input $\mathbf{x}$ is always considered in relation to a specified *uncertainty distribution*, defined by the density $u(\mathbf{x})$, and the appropriate marginal densities $u_i(x_i)$. What is represented by the uncertainty distribution changes depending upon the context. The canonical setup has that z is the response from a complicated physics or engineering simulation model, with tuning parameters $\mathbf{x}$, that is used to predict physical phenomena. In this situation, $u(\mathbf{x})$ represents the experimentalist's uncertainty about real–world values of $\mathbf{x}$. In optimization problems, the uncertainty distribution can be used to express prior information from experimentalists or modelers on where to look for solutions. Finally, in the case of observational systems (such as air-quality or smog levels), $u(\mathbf{x})$ may be an estimate of the density governing the natural occurrence of the $\mathbf{x}$ factors (e.g., air-pressure, temperature, wind, and cloud cover). In this setup, SA attempts to resolve the natural variability of z.

The most common notion of sensitivity is tied to the relationship between conditional and marginal variance for z. Specifically, variance–based methods decompose the variance of the objective function, with respect to the uncertainty distribution on the inputs, into variances of conditional expectations. These are a natural measure of the output association with specific sets of variables and provide a basis upon which the importance of individual inputs may be judged. The other common component of global SA is an accounting of the main effects for each input variable, $\mathbb{E}_{u_j}[z|x_j]$, which can be obtained as a by-product of the variance analysis.

Our variance–based approach to SA is a version of the method of Sobol', wherein a deterministic objective function is decomposed into summands of functions on lower dimensional subsets of the input space. Consider the function decomposition $f(x_1, \ldots, x_d) = f_0 + \sum_{j=1}^{d} f_j(x_j) + \sum_{1 \le i < j \le d} f_{ij}(x_j, x_i) + \ldots + f_{1,\ldots,d}(x_1, \ldots, x_d)$. When the response f is modeled as a stochastic process z conditional on inputs $\mathbf{x}$, we can develop a similar decomposition into the response distributions which arise when z has been marginalized over one subset of covariates and the complement of this subset is allowed to vary according to a marginalized uncertainty distribution. In particular, we can obtain the marginal conditional expectation $\mathbb{E}[z|\mathbf{x}_J = \{x_j : j \in J\}] = \int_{\mathbb{R}^{d-d_J}} \mathbb{E}[z|\mathbf{x}] u(\mathbf{x}) d\mathbf{x}_{-J}$, where $J = \{j_1, \ldots, j_{d_J}\}$ indicates a subset of input variables, $\mathbf{x}_{-j} = \{x_j : j \notin J\}$, and the marginal uncertainty density is given by $u_J(\mathbf{x}_J) = \int_{\mathbb{R}^{d-d_J}} u(\mathbf{x}) d\{x_i : i \notin J\}$. SA concerns the variability of $\mathbb{E}[z|\mathbf{x}_J]$ with respect to changes in $\mathbf{x}_J$ according to $u_J(\mathbf{x}_J)$ and, if u is such that the inputs are uncorrelated, the variance decomposition is available as

$$\mathrm{var}(\mathbb{E}[z|\mathbf{x}]) = \sum_{j=1}^{d} V_j + \sum_{1 \le i < j \le d} V_{ij} + \ldots + V_{1,\ldots,d}, \tag{2}$$

where $V_j = \mathrm{var}(\mathbb{E}[z|x_j])$, $V_{ij} = \mathrm{var}(\mathbb{E}[z|x_i, x_j]) - V_i - V_j$, and so on. Clearly, when the inputs are correlated this identity no longer holds (although a "less-than-or-equal-to" inequality is

always true). But it is useful to retain an intuitive interpretation of the V_J's as a portion of the overall marginal variance.

Our global SA will focus on the related sensitivity indices $S_J = V_J/\mathrm{var}(z)$ which, as can be seen in the above equation, will sum to one over all possible J and are bounded to $[0, 1]$. These S_J's provide a natural measure of the *importance* of a set J of inputs and serve as the basis for an elegant analysis of sensitivity. The `sens` function allows for easy calculation of two very important sensitivity indices associated with each input: the 1st order for the jth input variable,

$$S_j = \frac{\mathrm{var}\left(\mathbb{E}\left[z|x_j\right]\right)}{\mathrm{var}(z)}, \tag{3}$$

and the total sensitivity for input j,

$$T_j = \frac{\mathbb{E}\left[\mathrm{var}\left(z|\mathbf{x}_{-j}\right)\right]}{\mathrm{var}(z)}. \tag{4}$$

The 1st order indices measure the portion of variability that is due to variation in the main effects for each input variable, while the total effect indices measure the portion of variability that is due to total variation in each input. From the identity $\mathbb{E}\left[\mathrm{var}\left(z|\mathbf{x}_{-j}\right)\right] = \mathrm{var}(z) - \mathrm{var}\left(\mathbb{E}\left[z|\mathbf{x}_{-j}\right]\right)$, it can be seen that T_j measures the *residual* variability remaining after variability in all other inputs has been apportioned and that, for a deterministic response and uncorrelated input variables, $T_j = \sum_{J:j\in J} S_J$. This implies that the difference between T_j and S_j provides a measure of the variability in z due to interaction between input j and the other input variables. A large difference may lead the investigator to consider other sensitivity indices to determine where this interaction is most influential, and this is often a key aspect of the dimension–reduction that SA provides for optimization problems.

2.1 Monte Carlo integration for sensitivity indices

Due to the many integrals involved, estimation of the sensitivity indices is not straightforward. The influential paper by Oakley & O'Hagan [24] describes an empirical Bayes estimation procedure for the sensitivity indices, however some variability in the indices is lost due to plug-in estimation of GP model parameters and, more worryingly, the variance ratios are only possible in the form of a ratio of expected values. Marrel, et al. [19], provide a more complete analysis of the GP approach to this problem, but their methods remain restricted to estimation of the first order Sobol indices. Likelihood based approaches have also been proposed [38, 21]. The technique implemented in `tgp` is, in contrast, fully Bayesian and provides a complete accounting of the uncertainty involved. Briefly, at each iteration of an MCMC chain sampling from the treed GP posterior, output is predicted over a large (carefully chosen) set of input locations. Conditional on this predicted output, the sensitivity indices can be calculated via Monte Carlo integration. By conditioning on the predicted response (and working as though it were the observed response), we obtain a posterior sample of the indices, incorporating variability from both the integral estimation and uncertainty about the function output. In particular, the `sens` function includes a `model` argument which allows for SA based on any of the prediction models (the `b*` functions) in `tgp`.

Our Monte Carlo integration is based upon Saltelli's [32] efficient Latin hypercube sampling (LHS) scheme for estimation of both 1st order and total effect indices. We note that the estimation is only valid for uncorrelated inputs, such that $u(\mathbf{x}) = \prod_{j=1}^{d} u_j(x_j)$. The `sens` function only allows for uncertainty distributions of this type (in fact, the marginal distributions also need to be bounded), but this is a feature of nearly every "out-of-the-box" approach to SA. Studies which concern correlated inputs will inevitably require modeling for this correlation, whereas most regression models (including those in `tgp`) condition on the inputs and ignore the joint density for $\mathbf{x}$. Refer to the work of Saltelli & Tarantola [31] for an example of SA with correlated inputs.

We now briefly describe the integration scheme. The 2nd moment is a useful intermediate quantity in variance estimation, and we define

$$D_J = \mathbb{E}\left[\mathbb{E}^2\left[z|\mathbf{x}_J\right]\right] = \int_{\mathbb{R}^{d_J}} \mathbb{E}^2\left[z|\mathbf{x}_J\right] u_J(\mathbf{x}_J) d(\mathbf{x}_J).$$

Making use of an auxiliary variable,

$$\begin{aligned} D_J &= \int_{\mathbb{R}^{d_J}} \left[\int_{\mathbb{R}^{d_{-J}}} \mathbb{E}\left[z|\mathbf{x}_J, \mathbf{x}_{-J}\right] u_{-J}(\mathbf{x}_{-J}) d\mathbf{x}_{-J} \int_{\mathbb{R}^{d_{-J}}} \mathbb{E}\left[z|\mathbf{x}_J, \mathbf{x}'_{-J}\right] u_{-J}(\mathbf{x}'_{-J}) d\mathbf{x}'_{-J} \right] u_J(\mathbf{x}_J) \mathbf{x}_J \\ &= \int_{\mathbb{R}^{d+d_{-J}}} \mathbb{E}\left[z|\mathbf{x}_J, \mathbf{x}_{-J}\right] \mathbb{E}\left[z|\mathbf{x}_J, \mathbf{x}'_{-J}\right] u_{-J}(\mathbf{x}_{-J}) u_{-J}(\mathbf{x}'_{-J}) u_J(\mathbf{x}_J) d\mathbf{x} d\mathbf{x}'_J. \end{aligned}$$

Thus, in the case of independent inputs,

$$D_J = \int_{\mathbb{R}^{d+d_{-J}}} \mathbb{E}\left[z|\mathbf{x}\right] \mathbb{E}\left[z|\mathbf{x}_J, \mathbf{x}'_{-J}\right] u_{-J}(\mathbf{x}'_{-J}) u(\mathbf{x}) d\mathbf{x}'_{-J} d\mathbf{x}.$$

Note that at this point, if the inputs had been correlated, the integral would have been instead with respect to the joint density $u(\mathbf{x})u(\mathbf{x}'_{-J}|\mathbf{x}_J)$, leading to a more difficult integral estimation problem.

Recognizing that $S_j = (D_j - \mathbb{E}^2[z])/\mathrm{var}(z)$ and $T_j = 1 - ((D_{-j} - \mathbb{E}^2[z]))/\mathrm{var}(z)$, we need estimates of $\mathrm{var}(z)$, $\mathbb{E}^2[z]$, and $\{(D_j, D_{-j}) : j = 1, \ldots, d\}$ to calculate the sensitivity indices. Given a LHS M proportional to $u(\mathbf{x})$,

$$M = \begin{bmatrix} s_{1_1} & \cdots & s_{1_d} \\ & \vdots & \\ s_{m_1} & \cdots & s_{m_d} \end{bmatrix},$$

it is possible to estimate $\widehat{\mathbb{E}[z]} = \frac{1}{m}\sum_{k=1}^{m} \mathbb{E}[z|\mathbf{s}_k]$ and $\widehat{\mathrm{var}[z]} = \frac{1}{m}\mathbb{E}^T[z|M]\mathbb{E}[z|M] - \widehat{\mathbb{E}[z]}\widehat{\mathbb{E}[z]}$, where the convenient notation $\mathbb{E}[z|M]$ is taken to mean $[\mathbb{E}[z|\mathbf{s}_1] \cdots \mathbb{E}[z|\mathbf{s}_m]]^T$. All that remains is to estimate the D's. Define a second LHS M' proportional to u of the same size as M and say that N_J is M' with the J columns replaced by the corresponding columns of M. Hence,

$$N_j = \begin{bmatrix} s'_{1_1} \cdots s_{1_j} \cdots s'_{1_d} \\ \vdots \\ s'_{m_1} \cdots s_{m_j} \cdots s'_{m_d} \end{bmatrix} \quad \text{and} \quad N_{-j} = \begin{bmatrix} s_{1_1} \cdots s'_{1_j} \cdots s_{1_d} \\ \vdots \\ s_{m_1} \cdots s'_{m_j} \cdots s_{m_d} \end{bmatrix}.$$

The estimates are then $\hat{D}_j = \mathbb{E}^T[z|M]\mathbb{E}[z|N_j]/(m-1)$ and $\hat{D}_{-j} = \mathbb{E}^T[z|M']\mathbb{E}[z|N_j]/(m-1)$ $\approx \mathbb{E}^T[z|M]\mathbb{E}[z|N_{-j}]/(m-1)$. Along with the variance and expectation estimates, these can be plugged into equations for S_j and T_j in (3–4) to obtain $\hat{S}_j$ and $\hat{T}_j$. Note that Saltelli recommends the use of the alternative estimate $\widehat{\mathbb{E}^2[z]} = \frac{1}{n-1}\mathbb{E}^T[z|M]\mathbb{E}[z|M']$ in calculating 1^{st} order indices, as this brings the index closer to zero for non-influential variables. However, it has been our experience that these biased estimates can be unstable, and so `tgp` uses the standard $\widehat{\mathbb{E}^2[z]} = \widehat{\mathbb{E}[z]}\widehat{\mathbb{E}[z]}$ throughout. As a final point, we note that identical MCMC sampling-based integration schemes can be used to estimate other Sobol indices (e.g., second order, etc) for particular combinations of inputs, but that this would require customization of the `tgp` software.

The set of input locations which need to be evaluated for each calculation of the indices is $\{M, M', N_1, \ldots, N_d\}$, and if m is the sample size for the Monte Carlo estimate this scheme requires $m(d+2)$ function evaluations. Hence, at each MCMC iteration of the model fitting, the $m(d+2)$ locations are drawn randomly according the LHS scheme, creating a random prediction matrix, `XX`. By allowing random draws of the input locations, the Monte Carlo error of the integral estimates will be included in the posterior variability of the indices and the posterior moments will not be dependent upon any single estimation input set. Using predicted output over this input set, a single realization of the sensitivity indices is calculated through Saltelli's scheme. At the conclusion of the MCMC, we have a representative sample from the posterior for $\mathbf{S}$ and $\mathbf{T}$. The averages for these samples are unbiased estimates of the posterior mean, and the variability of the sample is representative of the complete uncertainty about model sensitivity.

Since a subset of the predictive locations (M and M') are actually a LHS proportional to the uncertainty distribution, we can also estimate the main effects at little extra computational cost. At each MCMC iteration, a one–dimensional nonparametric regression is fit through the scatterplot of $[s_{1_j}, \ldots, s_{m_j}, s'_{1_j}, \ldots, s'_{m_j}]$ vs. $[\mathbb{E}[z|M], \mathbb{E}[z|M']]$ for each of the $j = 1, \ldots, d$ input variables. The resultant regression estimate provides a realization of $\mathbb{E}[z|x_j]$ over a grid of x_j values, and therefore a posterior draw of the main effect curve. Thus, at the end of the MCMC, we have not only unbiased estimates of the main effects through posterior expectation, but also a full accounting of our uncertainty about the main effect curve. This technique is not very sensitive to the method of non-parametric regression, since $2m$ will typically represent a very large sample in one–dimension. The estimation in `tgp` uses a moving average with squared distance weights and a window containing the `span`$*2m$ nearest points (the `span` argument defaults to 0.3).

2.2 Examples

We illustrate the capabilities of the `sens` function by looking at the Friedman function considered earlier in this vignette. The function that describes the responses (Z), observed with standard Normal noise, has mean

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

A sensitivity analysis can be based upon any of the available regression models (e.g., `btlm`, `bgp`, or `btgp`); we choose to specify `model=btgpllm` for this example. The size of each LHS used in the integration scheme is specified through `nn.lhs`, such that this is equivalent to m in the above algorithm description. Thus the number of locations used for prediction—the size of the random `XX` prediction matrix—is `nn.lhs*(ncol(X)+2)`. In addition, the window for moving average estimation of the main effects is `span*2*nn.lhs` (independent of this, an `ngrid` argument with a default setting of `ngrid=100` dictates the number of grid points in each input dimension upon which main effects will be estimated).

```
> f <- friedman.1.data(250)
```

This function actually generates 10 covariates, the last five of which are completely uninfluential. We'll include one of these (x_6) to show what the sensitivity analysis looks like for unrelated variables.

```
> Xf <- f[, 1:6]
> Zf <- f$Y
> sf <- sens(X=Xf, Z=Zf, nn.lhs=600, model=bgpllm, verb=0)
```

The progress indicators printed to the screen (for `verb > 0`) are the same as would be obtained under the specified regression `model`—`bgpllm` in this case—so we suppress them here. All of the same options (e.g., `BTE`, `R`, etc.) apply, although if using the `trace` capabilities one should be aware that the `XX` matrix is changing throughout the MCMC. The `sens` function returns a `"tgp"`-class object, and all of the SA related material is included in the `sens` list within this object.

```
> names(sf$sens)

[1] "par"     "Xgrid"   "ZZ.mean" "ZZ.q1"   "ZZ.q2"   "S"
[7] "T"
```

The object provides the SA parameters (`par`), the grid of locations for main effect prediction (`Xgrid`), the mean and interval estimates for these main effects (`ZZ.mean`, `ZZ.q1`, and `ZZ.q2`), and full posterior via samples of the sensitivity indices (`S` and `T`).

The plot function for `"tgp"`-class objects now provides a variety of ways to visualize the results of a sensitivity analysis. This capability is accessed by specifying `layout="sens"` in the standard `plot` command. By default, the mean posterior main effects are plotted next to boxplot summaries of the posterior sample for each S_j and T_j index, as in Figure 5.

A further note on the role played by `nn.lhs`: As always, the quality of the regression model estimate depends on the length of the MCMC. But now, the quality of sensitivity analysis is directly influenced by the size of the LHS used for integral approximation; as with any Monte Carlo integration scheme, the sample size (i.e., `nn.lhs`) must increase with the dimensionality of the problem. In particular, it can be seen in the estimation procedure described above that the total sensitivity indices (the T_j's) are not forced to be non-negative.

```
> plot(sf, layout="sens", legendloc="topleft")
```

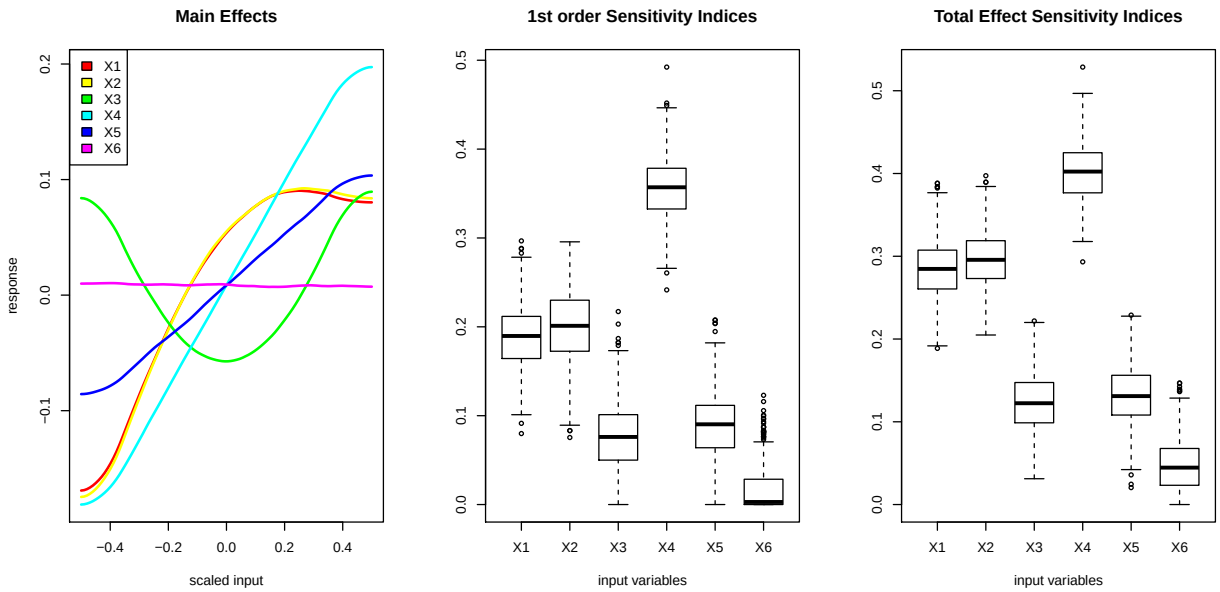

Figure 5: Full sensitivity analysis results for the Friedman function.

```
> par(mar=c(4,2,4,2), mfrow=c(2,3))
> plot(sf, layout="sens", maineff=t(1:6))
```

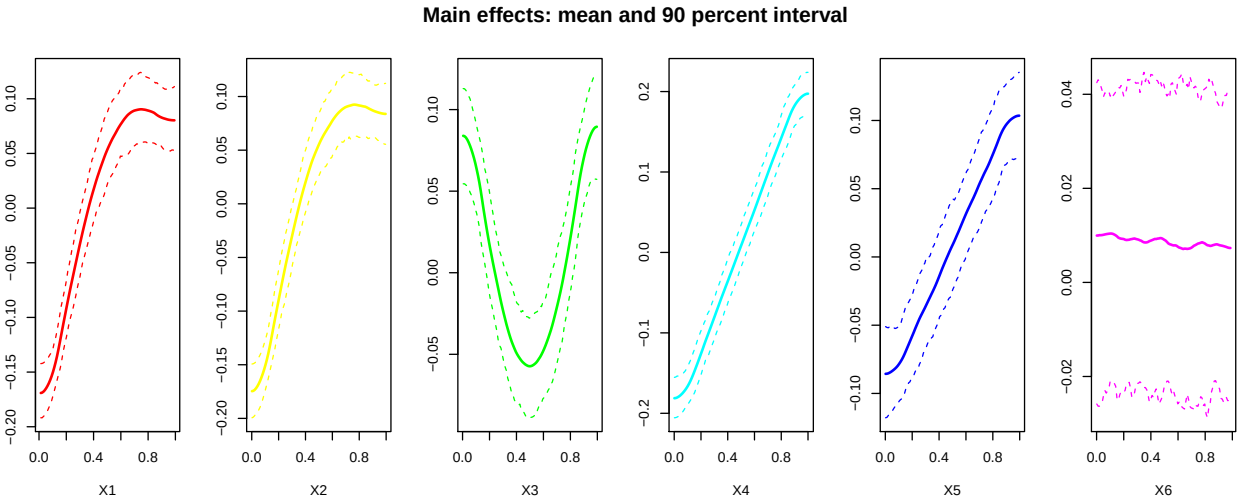

Figure 6: Friedman function main effects, with posterior 90% intervals.

If negative values occur it is necessary to increase `nn.lhs`. In any case, the `plot.tgp` function changes any of the negative values to zero for purposes of illustration.

The `maineff` argument can be used to plot either selected main effects (Figure 6), or just the sensitivity indices (Figure 7). Note that the posterior intervals shown in these plots

```
> plot(sf, layout="sens", maineff=FALSE)
```

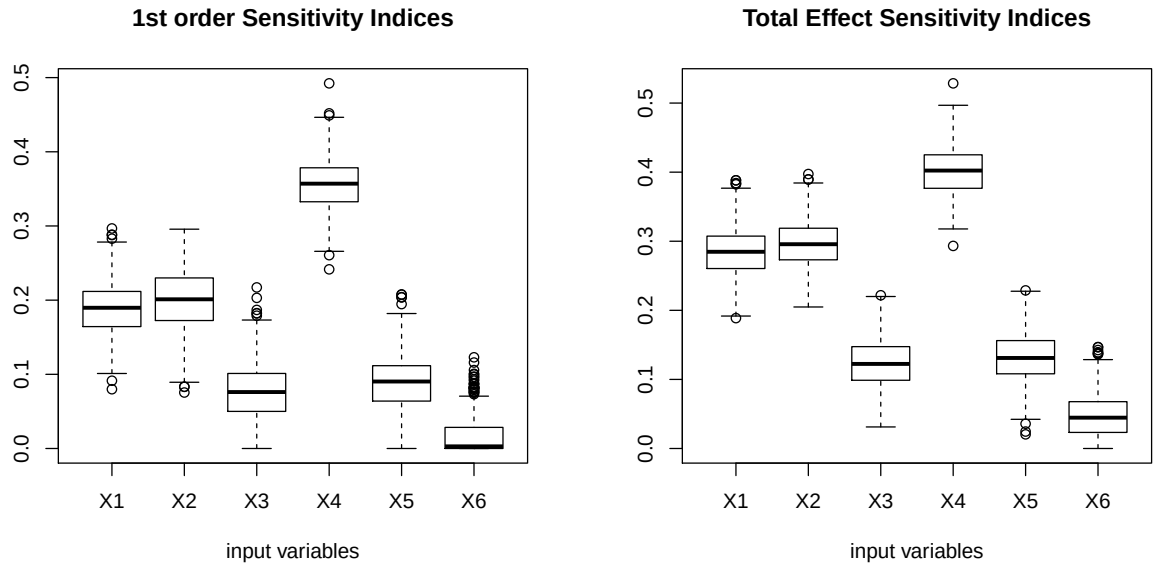

Figure 7: Sensitivity indices for the Friedman function.

represent uncertainty about both the function response and the integration estimates; this full quantification of uncertainty is not presently available in any alternative SA procedures. These plots may be compared to what we know about the Friedman function (refer to Eq. (5)) to evaluate the analysis. The main effects correspond to what we would expect: sine waves for x_1 and x_2, a parabola for x_3, and linear effects for x_4 and x_5. The sensitivity indices show x_1 and x_2 contributing roughly equivalent amounts of variation, while x_4 is relatively more influential than x_5. Full effect sensitivity indices for x_3, x_4, and x_5 are roughly the same as the first order indices, but (due to the interaction in the Friedman function) the sensitivity indices for the total effect of x_1 and x_2 are significantly larger than the corresponding first order indices. Finally, our SA is able to determine that x_6 is unrelated to the response.

This analysis assumes the default uncertainty distribution, which is uniform over the range of input data. In other scenarios, it is useful to specify an informative $u(\mathbf{x})$. In the `sens` function, properties of u are defined through the `rect`, `shape`, and `mode` arguments. To guarantee integrability of our indices, we have restricted ourselves to bounded uncertainty distributions. Hence, `rect` defines these bounds. In particular, this defines the domain from which the LHSs are to be taken. We then use independent scaled beta distributions, parameterized by the `shape` parameter and distribution `mode`, to define an informative uncertainty distribution over this domain.

As an example of sensitivity analysis under an informative uncertainty distribution, consider the `airquality` data available with the base distribution of R. This data set contains daily readings of mean ozone in parts per billion (*Ozone*), solar radiation (*Solar.R*), wind speed (*Wind*), and maximum temperature (*Temp*) for New York City, between May 1 and September 30, 1973. Suppose that we are interested in the sensitivity of air quality to nat-

ural changes in *Solar.R*, *Wind*, and *Temp*. For convenience, we will build our uncertainty distribution while assuming independence between these inputs. Hence, for each variable, the input uncertainty distribution will be a scaled beta with `shape=2`, and `mode` equal to the data mean.

```
> X <- airquality[,2:4]
> Z <- airquality$Ozone
> rect <- t(apply(X, 2, range, na.rm=TRUE))
> mode <- apply(X , 2, mean, na.rm=TRUE)
> shape <- rep(2,3)
```

LHS samples from the uncertainty distribution are shown in Figure (8)

```
> Udraw <- lhs(300, rect=rect, mode=mode, shape=shape)
> par(mfrow=c(1,3), mar=c(4,2,4,2))
> for(i in 1:3){
+   hist(Udraw[,i], breaks=10,xlab=names(X)[i],
+        main="",ylab="", border=grey(.9), col=8)
+ }
```

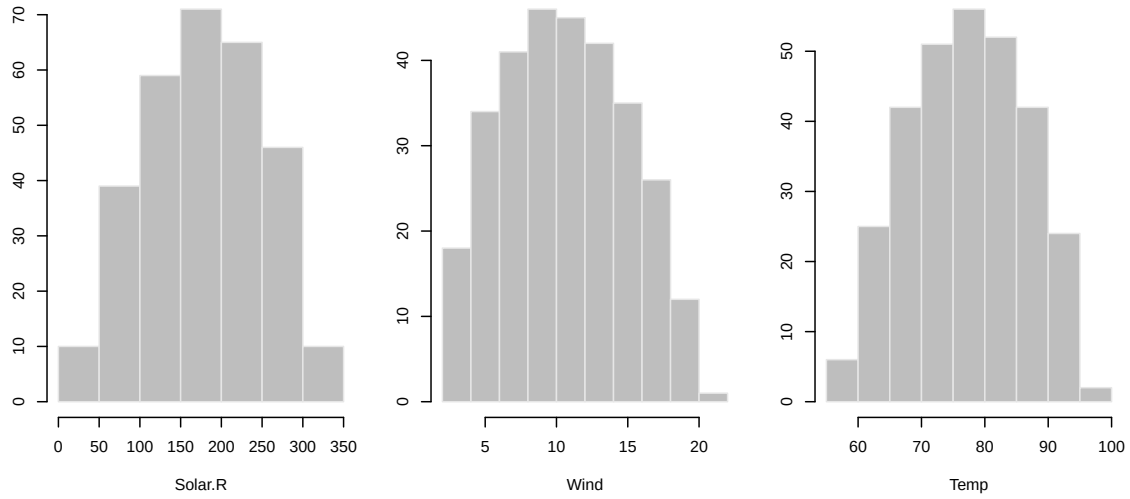

Figure 8: A sample from the marginal uncertainty distribution for the airquality data.

Due to missing data (discarded in the current version of `tgp`), we suppress warnings for the sensitivity analysis. We shall use the default `model=btgp`.

```
> s.air <- suppressWarnings(sens(X=X, Z=Z, nn.lhs=300, rect=rect,
+                                shape=shape, mode=mode, verb=0))
```

Figure (9) shows the results from this analysis.

```
> plot(s.air, layout="sens")
```

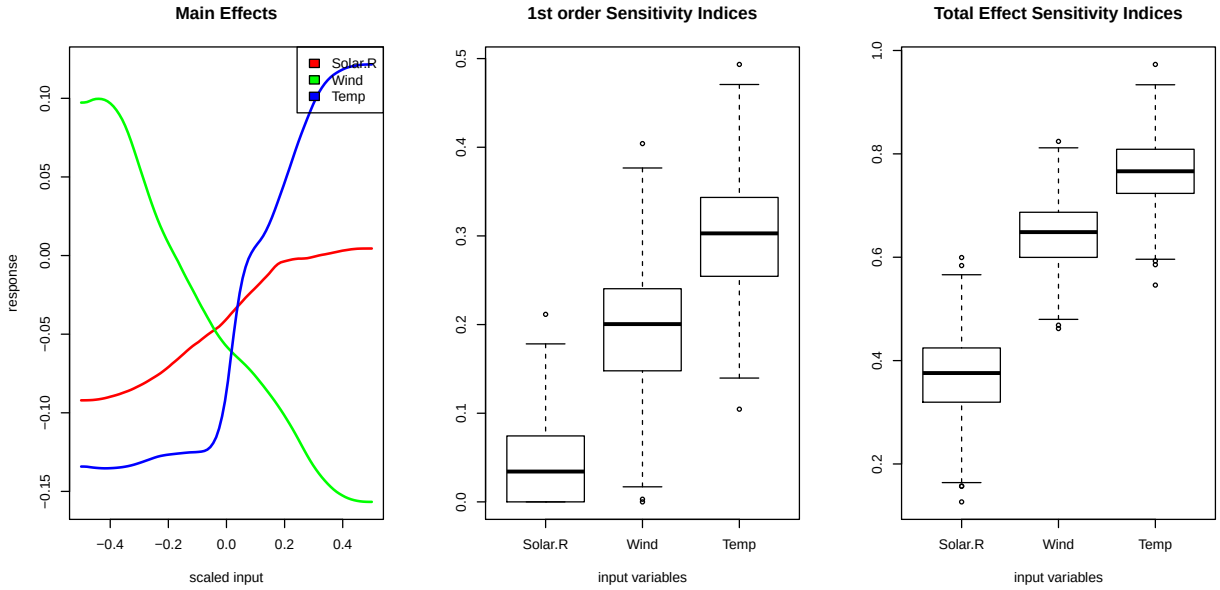

Figure 9: Sensitivity of NYC airquality to natural variation in wind, sun, and temperature.

Through use of `predict.tgp`, it is possible to quickly re-analyze with respect to a new uncertainty distribution without running new MCMC. We can, for example, look at sensitivity for air quality on only low–wind days. We thus alter the uncertainty distribution (assuming that things remain the same for the other variables)

```
> rect[2,] <- c(0,5)
> mode[2] <- 2
> shape[2] <- 2
```

and build a set of parameters `sens.p` with the `sens` function by setting `model=NULL`.

```
> sens.p <- suppressWarnings(sens(X=X,Z=Z,nn.lhs=300, model=NULL, rect=rect, shape=sha
```

Figures (9) and (10) both show total effect indices which are much larger than the respective first order sensitivities. As one would expect, the effect on airquality is manifest largely through an interaction between variables.

Finally, it is also possible to perform SA with binary covariates, included in the regression model as described in Section 1. In this case, the uncertainty distribution is naturally characterized by a Bernoulli density. Setting `shape[i]=0` informs `sens` that the relevant variable is binary (perhaps encoding a categorical input as in Section 1), and that the Bernoulli uncertainty distribution should be used. In this case, the `mode[i]` parameter dictates the probability parameter for the Bernoulli, and we must have `rect[i,] = c(0,1)`. As an example, we re-analyze the original air quality data with temperature included as an indicator variable (set to one if temperature > 79, the median, and zero otherwise).

```
> s.air2 <- predict(s.air, BTE=c(1,1000,1), sens.p=sens.p, verb=0)
> plot(s.air2, layout="sens")
```

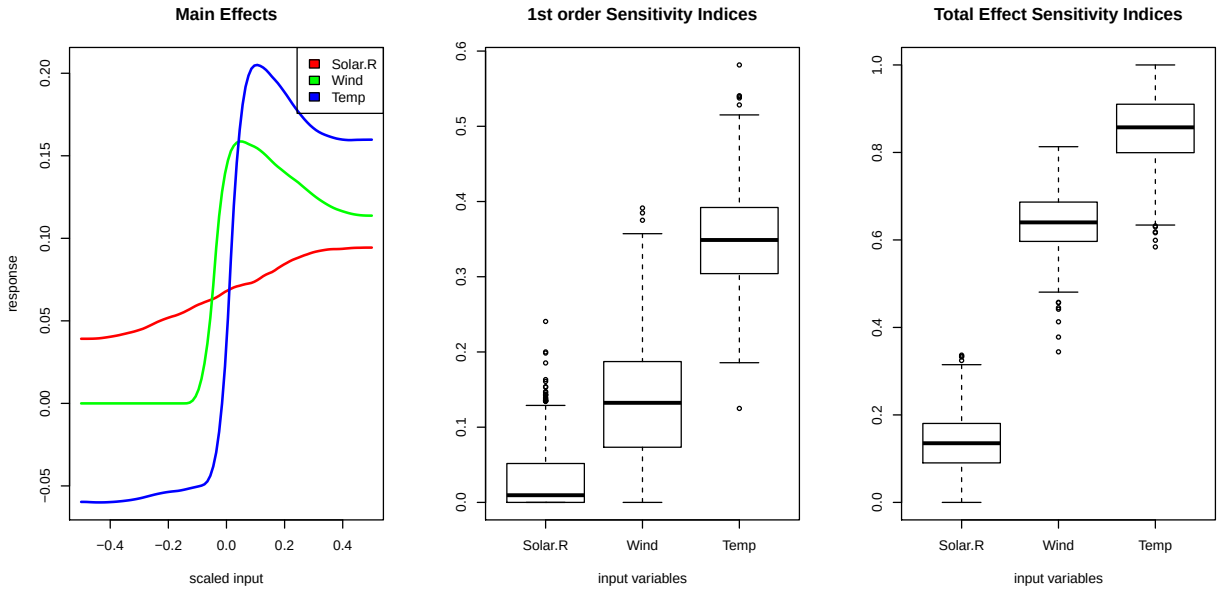

Figure 10: Air quality sensitivity on low-wind days.

```
> X$Temp[X$Temp >70] <- 1
> X$Temp[X$Temp >1] <- 0
> rect <- t(apply(X, 2, range, na.rm=TRUE))
> mode <- apply(X , 2, mean, na.rm=TRUE)
> shape <- c(2,2,0)
> s.air <- suppressWarnings(sens(X=X, Z=Z, nn.lhs=300, rect=rect,
+                                shape=shape, mode=mode, verb=0, basemax=2))
```

Figure (11) shows the results from this analysis.

3 Statistical search for optimization

There has been considerable recent interest in the use of statistically generated search patterns (i.e., locations of relatively likely optima) for optimization. A popular approach is to estimate a statistical (surrogate) model, and use it to design a set of well-chosen candidates for further evaluation by a direct optimization routine. Such statistically designed search patterns can be used either to direct the optimization completely (e.g., [16] or [28]) or to work in hybrid with local pattern search optimization (as in [36]). An bonus feature of the statistical surrogate approach is that it may be used to tackle problems of optimization under uncertainty, wherein the function being optimized is observed with noise. In this case the search is for input configurations which optimize the response with high probability. Direct-search methods would not apply in this scenario without modification. However, a sensible

```
> plot(s.air, layout="sens")
```

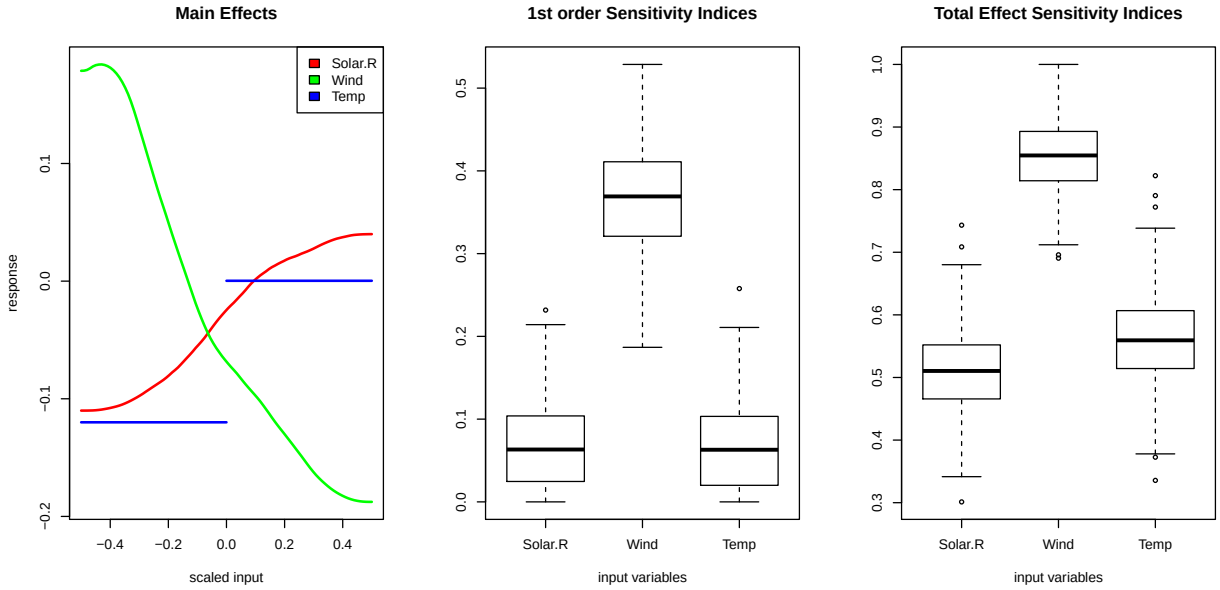

Figure 11: Sensitivity of NYC airquality to natural variation in wind, sun, and a binary temperature variable (for a threshold of 79 degrees).

hybrid could involve inverting the relationship between the two approaches so that direct-search is used on deterministic predictive surfaces from the statistical surrogate model. This search can be used to find promising candidates to compliment space-filling ones at which some statistical improvement criterion is evaluated.

Towards situating `tgp` as a promising statistical surrogate model for optimization (in both contexts) the approach developed by Taddy, et al. [36], has been implemented to produce a list of input locations that is ordered by a measure of the potential for new optima. The procedure uses samples from the posterior predictive distribution of treed GP regression models to estimate improvement statistics and build an ordered list of search locations which maximize expected improvement. The single location improvement is defined $I(\mathbf{x}) = \max\{f_{min} - f(\mathbf{x}), 0\}$, where f_{min} is the minimum evaluated response in the search (refer to [33] for extensive discussion on general improvement statistics and initial vignette [11] for details of a base implementation in `tgp`). Thus, a high improvement corresponds to an input location that is expected to be much lower than the current minimum. The criterion is easily changed to a search for maximum values through negation of the response. The improvement is always non-negative, as points which do not turn out to be new minimum points still provide valuable information about the output surface. Thus, in the expectation, candidate locations will be rewarded for high response uncertainty (indicating a poorly explored region of the input space), as well as for low mean predicted response. Our `tgp` generated search pattern will consist of m locations that recursively maximize (over a discrete candidate set) a sequential version of the expected multi-location improvement developed by

Schonlau, et al. [33], defined as $\mathbb{E}\left[I^g(\mathbf{x}_1,\ldots,\mathbf{x}_m)\right]$ where

$$I^g(\mathbf{x}_1,\ldots,\mathbf{x}_m) = \left(\max\{(f_{min}-f(\mathbf{x}_1)),\ldots,(f_{min}-f(\mathbf{x}_m)),0\}\right)^g . \tag{6}$$

Increasing $g \in \{0,1,2,3,\ldots\}$ increases the global scope of the criteria by rewarding in the expectation extra variability at $\mathbf{x}$. For example, $g=0$ leads to $\mathbb{E}[I^0(\mathbf{x})] = \Pr(I(\mathbf{x})>0)$ (assuming the convention $0^0=0$), $g=1$ yields the standard statistic, and $g=2$ explicitly rewards the improvement variance since $\mathbb{E}[I^2(\mathbf{x})] = \mathrm{var}[I(\mathbf{x})] + \mathbb{E}[I(\mathbf{x})]^2$. For further discussion on the role of g, see [33] .

Finding the maximum expectation of (6) is practically impossible for the full posterior distribution of $I^g(\mathbf{x}_1,\ldots,\mathbf{x}_m)$, and would require conditioning on a single fit for the model parameters (for example, static imputation of predictive GP means can be used to recursively build the improvement set [10]). However, `tgp` just seeks to maximize over a discrete list of predictive locations. In fact, the default is to return an ordering for the entire `XX` matrix, thus defining a ranking of predictive locations by order of decreasing expected improvement. There is no restriction on the form for `XX`.[1] The structure of this scheme will dictate the form for `XX`. If it is the case that we seek simply to explore the input space and map a list of potential locations for improvement, using LHS to choose `XX` will suffice.

The discretization of decision space allows for a fast iterative solution to the optimization of $\mathbb{E}\left[I^g(\mathbf{x}_1,\ldots,\mathbf{x}_m)\right]$. This begins with evaluation of the simple improvement $I^g(\tilde{\mathbf{x}}_i)$ over $\tilde{\mathbf{x}}_i \in \tilde{\mathbf{X}}$ at each of $T =$ `BTE[2]-BTE[1]` MCMC iterations (each corresponding to a single posterior realization of `tgp` parameters and predicted response after burn-in) to obtain the posterior sample

$$\mathcal{I} = \left\{ \begin{array}{ccc} I^g(\tilde{\mathbf{x}}_1)_1 & \ldots & I^g(\tilde{\mathbf{x}}_m)_1 \\ & \vdots & \\ I^g(\tilde{\mathbf{x}}_1)_T & \ldots & I^g(\tilde{\mathbf{x}}_m)_T \end{array} \right\}.$$

Recall that in `tgp` parlance, and as input to the `b*` functions: $\tilde{\mathbf{X}} \equiv$ `XX`.

We then proceed iteratively to build an *ordered* collection of m locations according to an iteratively refined improvement: Designate $\mathbf{x}_1 = \mathrm{argmax}_{\tilde{\mathbf{x}}\in\tilde{\mathbf{X}}}\mathbb{E}\left[I^g(\tilde{\mathbf{x}})\right]$, and for $j=2,\ldots,m$, given that $\mathbf{x}_1,\ldots,\mathbf{x}_{j-1}$ are already included in the collection, the next member is

$$\begin{aligned} \mathbf{x}_j &= \mathrm{argmax}_{\tilde{\mathbf{x}}\in\tilde{\mathbf{X}}}\mathbb{E}\left[\max\{I^g(\mathbf{x}_1,\ldots,\mathbf{x}_{j-1}), I^g(\tilde{\mathbf{x}})\}\right] \\ &= \mathrm{argmax}_{\tilde{\mathbf{x}}\in\tilde{\mathbf{X}}}\mathbb{E}[(\max\{(f_{min}-f(\mathbf{x}_1)),\ldots,(f_{min}-f(\mathbf{x}_{j-1})),(f_{min}-f(\tilde{\mathbf{x}})),0\})^g] \\ &= \mathrm{argmax}_{\tilde{\mathbf{x}}\in\tilde{\mathbf{X}}}\mathbb{E}\left[I^g(\mathbf{x}_1,\ldots,\mathbf{x}_{j-1},\tilde{\mathbf{x}})\right]. \end{aligned}$$

Thus, after each j^{th} additional point is added to the set, we have the maximum expected j–location improvement conditional on the first $j-1$ locations. This is not necessarily the unconditionally maximal expected j–location improvement; instead, point $\mathbf{x}_j$ is the location

[1] A full optimization routine would require that the search pattern is placed within an algorithm iterating towards convergence, as in [36]. However, we concentrate here on the statistical problem of choosing the next samples optimally. We shall touch on issues of convergence in Section 3.2 where we describe a skeleton scheme for optimization extending R's internal `optim` functionality.

which will cause the greatest increase in expected improvement over the given $(j-1)$–location expected improvement.

The posterior sample $\mathcal{I}$ acts as a discrete approximation to the true posterior distribution for improvement at locations within the candidate set `XX`. Based upon this approximation, iterative selection of the point set is possible without any re-fitting of the `tgp` model. Conditional on the inclusion of $\tilde{\mathbf{x}}_{i_1}, \ldots, \tilde{\mathbf{x}}_{i_{l-1}}$ in the collection, a posterior sample of the l–location improvement statistics is calculated as

$$\mathcal{I}_l = \left\{ \begin{array}{ccc} I^g(\tilde{\mathbf{x}}_{i_1}, \ldots, \tilde{\mathbf{x}}_{i_{l-1}}, \tilde{\mathbf{x}}_1)_1 & \ldots & I^g(\tilde{\mathbf{x}}_{i_1}, \ldots, \tilde{\mathbf{x}}_{i_{l-1}}, \tilde{\mathbf{x}}_m)_1 \\ & \vdots & \\ I^g(\tilde{\mathbf{x}}_{i_1}, \ldots, \tilde{\mathbf{x}}_{i_{l-1}}, \tilde{x}_1)_T & \ldots & I^g(\tilde{\mathbf{x}}_{i_1}, \ldots, \tilde{\mathbf{x}}_{i_{l-1}}, \tilde{\mathbf{x}}_m)_T \end{array} \right\},$$

where the element in the t^{th} row and j^{th} column of this matrix is calculated as $\max\{I^g(\tilde{\mathbf{x}}_{i_1}, \ldots, \tilde{\mathbf{x}}_{i_{l-1}})_t, I^g(\tilde{\mathbf{x}}_j)_t\}$ and the l^{th} location included in the collection corresponds to the column of this matrix with maximum average. Since the multi-location improvement is always at least as high as the improvement at any subset of those locations, the same points will not be chosen twice for inclusion. In practice, very few iterations (about 10% of the total candidate size under the default inference and regression model(s)) through this ordering process can be performed before the iteratively updated improvement statistics become essentially zero. Increasing the number of MCMC iterations (`BTE[2]-BTE[1]`) can mitigate this to a large extent.[2] We refer the reader to [36] for further details on this approach to multi-location improvement search.

3.1 A simple example

We shall use the Rosenbrock function to illustrate the production of an ordered collection of (possible) adaptive samples to maximize the expected improvement within `tgp`. Specifically, the two dimensional Rosenbrock function is defined as

```
> rosenbrock <- function(x){
+   x <- matrix(x, ncol=2)
+   100*(x[,1]^2 - x[,2])^2 + (x[,1] - 1)^2
+ }
```

and we shall bound the search space for adaptive samples to the rectangle: $-1 \leq x_i \leq 5$ for $i = 1, 2$. The single global minimum of the Rosenbrock function is at $(1, 1)$.

```
> rosenbrock(c(1,1))

[1] 0
```

[2] Once a zero (maximal) iterative improvement is attained the rest of the ranking is essentially arbitrary, at which point `tgp` cuts off the process prematurely.

This function involves a long steep valley with a gradually sloping floor, and is considered to be a difficult problem for local optimization routines.

We begin by drawing an LHS of 40 input locations within the bounding rectangle, and evaluating the function at these locations.

```
> rect <- cbind(c(-1,-1),c(5,5))
> X <- lhs(40, rect)
> Z <- rosenbrock(X)
```

We will fit a `bgp` model to this data to predict the Rosenbrock response at unobserved (candidate) input locations in `XX`. The `improv` argument may be used to obtain an ordered list of places where we should be looking for new minima. In particular, specifying `improv=c(1,10)` will return the 10 locations which maximize the iterative multi-location expected improvement function, with $g = 1$ (i.e., Eq. (6)). Note that `improv=TRUE` is also possible, in which case `g` defaults to one and the entire list of locations is ranked. Our candidate set is just a space filling LHS design. In other situations, it may be useful to build an informative LHS design (i.e., to specify `shape` and `mode` arguments for the `lhs` function) to reflect what is already known about the location of optima.

```
> XX <- lhs(200, rect)
> rfit <- bgp(X,Z,XX,improv=c(1,10), verb=0)
```

Upon return, the `"tgp"`-class object `rfit` includes the matrix `improv`, which is a list of the expected single location improvement for the 200 `XX` locations, and the top 10 ranks. Note that the `rank`s for those points which are not included in the top 10 are set to `nrow(XX)`=200. Here are the top 10:

```
> cbind(rfit$improv,XX)[rfit$improv$rank <= 10,]
```

```
          improv rank           1           2
12  0.0006620366    5  0.04339625  0.09489153
20  0.0006456795    3  1.31313812  1.87566387
21  0.0006022919    6  1.60593438  2.38150541
110 0.0006571810    2  0.59953907  0.26334479
119 0.0006140808    7  1.61718015  2.39090831
143 0.0005395790    9 -0.84254284  1.04683957
154 0.0006724039    1  0.13792447 -0.01324855
164 0.0005741945    8  0.05915976 -0.27341950
182 0.0005802249   10  2.22019217  4.76719757
200 0.0006329528    4  2.04231115  3.99968412
```

This iterative algorithm may produce ranks that differ significantly from a straightforward ordering of expected improvement. This leads to a list that better explores the input space, since the expected improvement is naturally balanced against a desire to search the domain.

We plot the results with the usual function, by setting `as="improv"`, in Figure 12. The

```
> plot(rfit, as="improv")
```

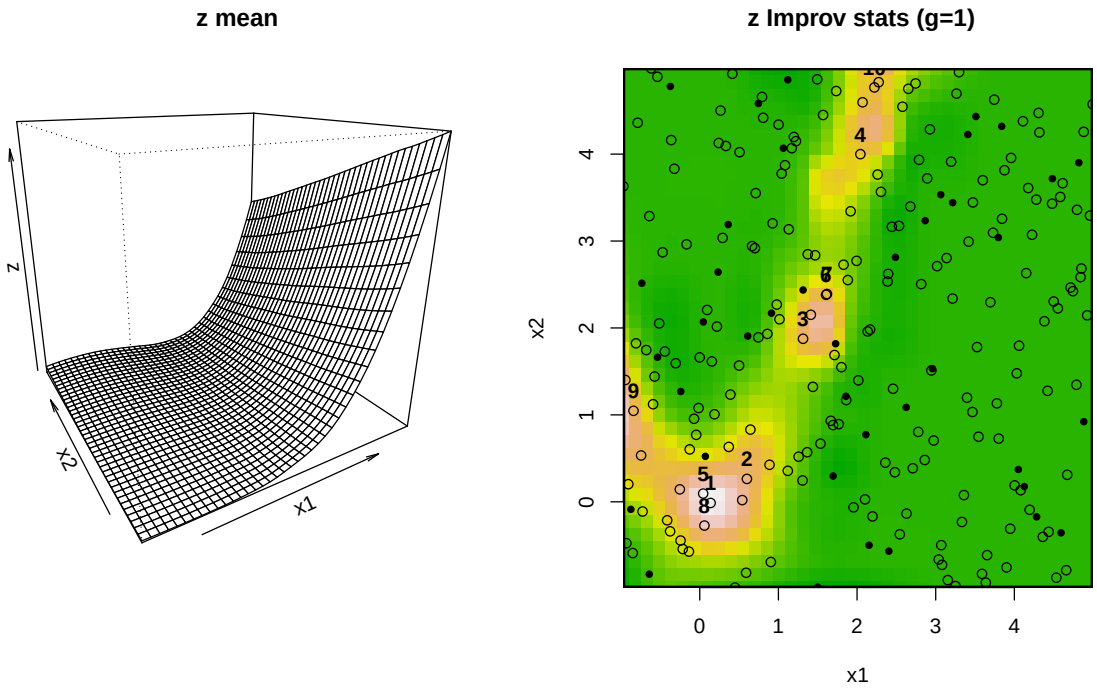

Figure 12: The *left* panel shows the mean predicted Rosenbrock function response, and on the *right* we have expected single location improvement with the top 10 points (labelled by rank) plotted on top.

banana–shaped region of higher expected improvement corresponds to the true valley floor for the Rosenbrock function, indicating the that `bgp` model is doing a good job of prediction. Also, we note that the ordered input points are well dispersed throughout the valley—a very desirable property for adaptive sampling candidates.

It is straightforward, with `predict.tgp`, to obtain a new ordering for the more global `g=5` (or any new `g`). Figure 13 shows a more diffuse expected improvement surface and a substantially different point ordering. In practice, we have found that `g=2` provides a good compromise between local and global search.

3.2 A skeleton optimization scheme

The capabilities outlined above are useful in their own right, as a search list or candidate set ranked by expected improvement gain provides concrete information about potential optima. However, a full optimization framework requires that the production of these sets of search locations are nested within an iterative search scheme. The approach taken by Taddy, et al. [36], achieves this by taking the `tgp` generated sets of locations and using them to augment a local optimization search algorithm. In this way, the authors are able to achieve robust solutions which balance the convergence properties of the local methods with the global scope provided by `tgp`. Indeed, any optimization routine capable of evaluating points provided by

```
> rfit2 <- predict(rfit, XX=XX, BTE=c(1,1000,1), improv=c(5,20), verb=0)
> plot(rfit2, layout="as", as="improv")
```

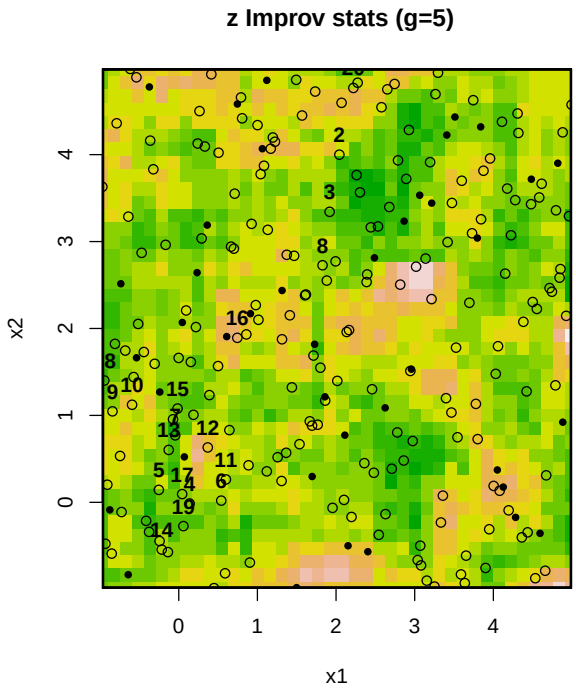

Figure 13: The expected improvement surface and top 20 ordered locations, for `g=5`.

an outside source could benefit from a `tgp` generated list of search locations.

In the absence of this sort of formal hybrid search algorithm, it is still possible to devise robust optimization algorithms based around `tgp`. A basic algorithm is as follows: first, use a LHS to explore the input space (see the `lhs` function included in `tgp`). Repeatedly fit one of the `b*` models with `improv!=FALSE` to the evaluated iterates to produce a search set, then evaluate the objective function over this search set, as described earlier. Then evaluate the objective function over the highest ranked locations in the search set. Continue until you are confident that the search has narrowed to a neighborhood around the true optimum (a good indicator of this is when all of the top-ranked points are in the same area). At this point, the optimization may be completed by `optim`, R's general purpose local optimization algorithm in order to guarentee convergence. The `optim` routine may be initialized to the best input location (i.e. corresponding the most optimal function evaluation) found thus far by `tgp`.

Note that this approach is actually an extreme version of a template proposed by Taddy, et al. [36], where the influence of global (i.e. `tgp`) search is downweighted over time rather than cut off. In either case, a drawback to such approaches is that they do not apply when the function being optimized is deterministic. An alternative scheme is to employ both `tgp` search and a local optimization at each iteration. The idea is that a mix of local and global information is provided throughout the entire optimization, but with an added twist.

Rather than apply `optim` on the stochastic function directly, which would not converge due to the noise, it can be applied on a deterministic (MAP) kriging surface provided by `tgp`. The local optima obtained can be used to augment the candidate set of locations where the improvement statistic is gathered—which would otherwise be simple LHS. That way the search pattern produced on output is likely to have a candidate with high improvement.

To fix ideas, and for the sake of demonstration, the `tgp` package includes a skeleton function for performing a single iteration in the derivative–free optimization of noisy black–box functions. The function is called `optim.step.tgp`, and the name is intended to emphasize that it performs a single step in an optimization by trading off local `optim`–based search of `tgp` predictive (kriging surrogate) surfaces, with the expected posterior improvement. In other words, it is loosely based on some the techniques alluded to above, but is designed to be augmented/adjusted as needed. Given N pairs of inputs and responses $(\mathbf{X}, \mathbf{Z})$, `optim.step.tgp` suggests new points at which the function being optimized should be evaluated. It also returns information that can be used to assess convergence. An outline follows.

The `optim.step.tgp` function begins by constructing a set of candidate locations, either as a space filling LHS over the input space (the default) or from a treed D–optimal design, based on a previously obtained `"tgp"`-class model. R's `optim` command is used on the MAP predictive surface contained within the object to obtain an estimate of the current best guess $\mathbf{x}$-location of the optimal solution. A standalone subroutine called `optim.ptgpf` is provided for this specific task, to be used within `optim.step.tgp` or otherwise. Within `optim.step.tgp`, `optim.ptgpf` is initialized with the data location currently predicted to be the best guess of the minimum. The optimal x-location found is then added into the set of candidates as it is likely that the expected improvement would be high there.

Then, a new `"tgp"`-class object is obtained by applying a `b*` function to $(\mathbf{X}, \mathbf{Z})$ whilst sampling from the posterior distribution of the improvement statistic. The best one, two, or several locations with highest improvement ranks are suggested for addition into the design. The values of the maximum improvement statistic are also returned in order to track progress in future iterations. The `"tgp"`-class object returned is used to construct candidates and initialize the `optim.ptgpf` function in future rounds.

To illustrate, consider the 2-d exponential data from the initial vignette [11] as our noisy function f.

```
> f <- function(x) { exp2d.Z(x)$Z }
```

Recall that this data is characterized by a mean value of

$$f(\mathbf{x}) = x_1 \exp(-x_1^2 - x_2^2)$$

which is observed with a small amount of Gaussian noise (with sd = 0.001). Elementary calculus gives that the minimum of f is obtained at $\mathbf{x} = (-\sqrt{1/2}, 0)$.

The `optim.step.tgp` function requires that the search domain be defined by a bounding rectangle, and we require an initial design to start things off. Here we shall use $[-2, 6]^2$ with an LHS design therein.

```
> rect <- rbind(c(-2,6), c(-2,6))
> X <- lhs(20, rect)
> Z <- f(X)
```

The following code proceeds with several rounds of sequential design towards finding the minimum of `f`.

```
> out <- progress <- NULL
> for(i in 1:20) {
+
+   ## get recommendations for the next point to sample
+   out <- optim.step.tgp(f, X=X, Z=Z, rect=rect, prev=out, verb=0)
+
+   ## add in the inputs, and newly sampled outputs
+   X <- rbind(X, out$X)
+   Z <- c(Z, f(out$X))
+
+   ## keep track of progress and best optimum
+   progress <- rbind(progress, out$progress)
+ }
```

The `progress` can be tracked through the rows of a `data.frame`, as constructed above, containing a listing of the input location of the current best guess of the minimum for each round, together with the value of the objective at that point, as well as the maximum of the improvement statistic. In addition to printing this data to the screen, plots such as the ones in Figure 14 can be valuable for assessing convergence. As can be seen in the figure, the final iteration gives an $\mathbf{x}$-value that is very close to the correct result, and is (in some loose sense) close to convergence.

```
> out$progress[1:2]

          x1           x2
1 -0.7053245 -0.002120479
```

As mentioned above, if it is known that the function evaluations are deterministic then, at any time, R's `optim` routine can be invoked—perhaps initialized by the $\boldsymbol{x}$-location in `out$progress`—and convergence to a local optimum thus guaranteed. Otherwise, the quantities in `out$progress` will converge, in some sense, as long as the number of MCMC rounds used in each round, above, ($T =$ `BTE[2]-BTE[1]`) tends to infinity. Such arguments to the `b*` functions can be set via the ellipses (`...`) arguments to `optim.step.tgp`.[3] A heuristic stopping criterion can be based on the maximum improvement statistic obtained in each

[3]This runs contrary to how the ellipses are used by `optim` in order to specify static arguments to `f`. If setting static arguments to `f` is required within `optim.step.tgp`, then they must be set in advance by adjusting the default arguments via `formals`.

```
> par(mfrow=c(1,2))
> matplot(progress[,1:2], main="x progress",
+         xlab="rounds", ylab="x[,1:2]", type="l", lwd=2)
> legend("topright", c("x1", "x2"), lwd=2, col=1:2, lty=1:2)
> plot(log(progress$improv), type="l", main="max log improv",
+      xlab="rounds", ylab="max log(improv)")
```

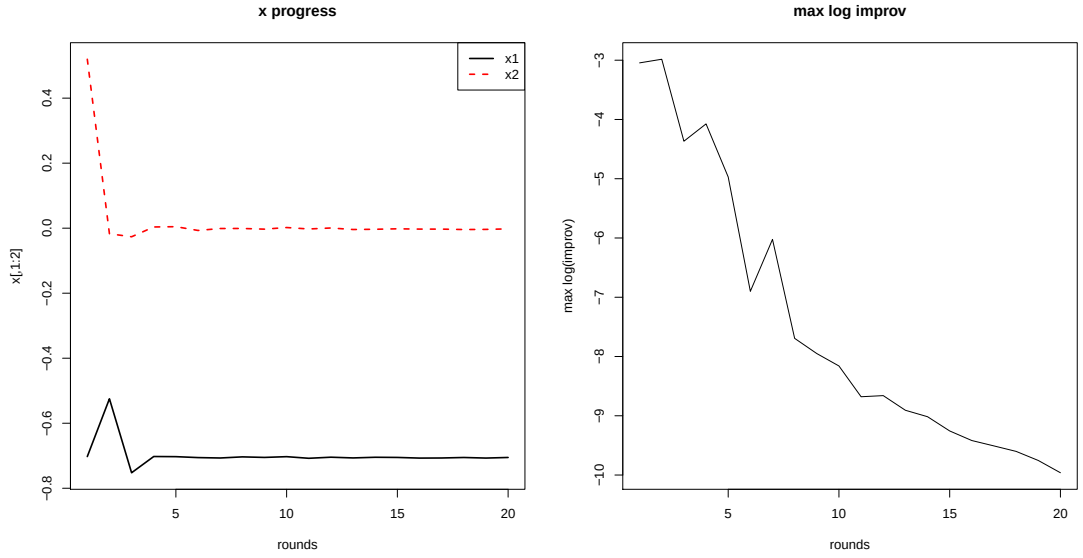

Figure 14: Progress in iterations of `optim.step.tgp` shown by tracking the $\mathbf{x}$–locations of the best guess of the minimum (*left*) and the logarithm of the maximum of the improvement statistics at the candidate locations (*right*)

round as long as the candidate locations become dense in the region as $T \to \infty$. This can be adjusted by increasing the `NN` argument to `optim.step.tgp`.

The internal use of `optim` within `optim.step.tgp` on the posterior predictive (kriging surrogate) surface via `optim.ptgpf` may proceed with any of the usual method arguments. I.e.,

```
> formals(optim)$method

c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN", "Brent")
```

however the default ordering is switched in `optim.ptgpf` and includes one extra method.

```
> formals(optim.ptgpf)$method

c("L-BFGS-B", "Nelder-Mead", "BFGS", "CG", "SANN", "optimize")
```

Placing `"L-BFGS-B"` in the default position is sensible since this method enforces a rectangle of constraints as specified by `rect`. This guarentees that the additional candidate found by

`optim.ptfpf` will be valid. However, the other `optim` methods generally work well despite that they do not enforce this constraint. The final method, `"optimize"`, applies only when the inputs to `f` are 1-d. In this case, the documentation for `optim` suggests using the `optimize` function instead.

4 Importance tempering

It is well–known that MCMC inference in Bayesian treed methods suffers from poor mixing. For example, Chipman et al. [2, 3] recommend periodically restarting the MCMC to avoid chains becoming stuck in local modes of the posterior distribution (particularly in tree space). The treed GP models are or no exception, although it is worth remarking that using flexible GP models at the leaves of the tree typically results in shallower trees, and thus less pathalogical mixing in tree space. Version 1.x provided some crude tools to help mitigate the effects of poor mixing in tree space. For example, the `R` argument to the `b*` functions facilitates the restarts suggested by Chipman et al.

A modern Monte Carlo technique for dealing with poor mixing in Markov chain methods is to employ *tempering* to flatten the peaks and raise the troughs in the posterior distribution so that movements between modes is more fluid. One such method, called *simulated tempering* (ST) [9], is essentially the MCMC analogue of the popular simulated annealing algorithm for optimization. The ST algorithm helps obtain samples from a multimodal density $\pi(\theta)$ where standard methods, such as Metropolis–Hastings (MH) [20, 13] and Gibbs Sampling (GS) [7], fail.

As will be shown in our examples, ST can guard against becoming stuck in local modes of the `tgp` posterior by encouraging better mixing *between modes* via in increase in the acceptance rate of tree modification proposals, particularly *prunes*. However, as we will see, ST suffers from inefficiency because it discards the lions share of the samples it collects. The discarded samples can be recycled if they are given appropriate importance sampling (IS) [18] weights. These weights, if combined carefully, can be used to construct meta-estimators of expectations under the `tgp` posterior that have much lower variance compared to ST alone. This combined application of ST and IT is dubbed *importance tempering* [12].

4.1 Simulated Tempering and related methods

ST is an application of the MH algorithm on the product space of parameters and inverse temperatures $k \in [0, 1]$. That is, ST uses MH to sample from the joint chain $\pi(\theta, k) \propto \pi(\theta)^k p(k)$. The inverse temperature is allowed to take on a discrete set of values $k \in \{k_1, \ldots, k_m : k_1 = 1,\ k_i > k_{i+1} \geq 0\}$, called the *temperature ladder*. Typically, ST calls for sampling $(\theta, k)^{(t+1)}$ by first updating $\theta^{(t+1)}$ conditional on $k^{(t)}$ and (possibly) on $\theta^{(t)}$, using MH or GS. Then, for a proposed $k' \sim q(k^{(t)} \rightarrow k')$, usually giving equal probability to the nearest inverse temperatures greater and less than $k^{(t)}$, an acceptance ratio is calculated:

$$A^{(t+1)} = \frac{\pi(\theta^{(t+1)})^{k'} p(k') q(k' \rightarrow k^{(t)})}{\pi(\theta^{(t+1)})^{k^{(t)}} p(k^{(t)}) q(k^{(t)} \rightarrow k')}.$$

Finally, $k^{(t+1)}$ is determined according to the MH accept/reject rule: set $k^{(t+1)} = k'$ with probability $\alpha^{(t+1)} = \min\{1, A^{(t+1)}\}$, or $k^{(t+1)} = k^{(t)}$ otherwise. Standard theory for MH and GS gives that samples from the marginals π_{k_i} can be obtained by collecting samples $\theta^{(t)}$ where $k^{(t)} = k_i$. Samples from $\pi(\theta)$ are obtained when $k^{(t)} = 1$.

The success of ST depends crucially on the ability of the Markov chain frequently to: (a) visit high temperatures (low k) where the probability of escaping local modes is increased; (b) visit $k = 1$ to obtain samples from π. The algorithm can be tuned by: (i.) adjusting the number and location of the rungs of the temperature ladder; or (ii.) setting the pseudo-prior $p(k)$ for the inverse temperature.

Geyer & Thompson [9] give ways of adjusting the spacing of the rungs of the ladder so that the ST algorithm achieves between–temperature acceptance rates of 20–40%. More recently, authors have preferred to rely on defaults, e.g.,

$$k_i = \begin{cases} (1+\Delta_k)^{1-i} & \text{geometric spacing} \\ \{1+\Delta_k(i-1)\}^{-1} & \text{harmonic spacing} \end{cases} \qquad i = 1, \ldots, m. \tag{7}$$

Motivation for such default spacings is outlined by Liu [18]. Geometric spacing, or uniform spacing of $\log(k_i)$, is also advocated by Neal [22, 23] to encourage the Markov chain to rapidly traverse the breadth of the temperature ladder. Harmonic spacing is more often used by a related method called Metropolis coupled Markov chain Monte Carlo (MC^3) [8]. Both defaults are implemented in the `tgp` package, through the provided `default.itemps` function. A new "sigmoidal" option is also implemented, as discussed below. The rate parameter $\Delta_k > 0$ can be problem specific. Rather than work with Δ_k the `default.itemps` function allows the ladder to be specified via m and the hottest temperature k_m, thus fixing Δ_k implicitly. I.e., for the geometric ladder $\Delta_k = (k_m)^{1/(1-m)} - 1$, and for the harmonic ladder $\Delta_k = \frac{(k_m)^{-1}-1}{m-1}$.

A sigmoidal ladder can provide a higher concentration of temperatures near $k = 1$ without sacrificing the other nice properties of the geometric and harmonic ladders. It is specified by first situating m indices $j_i \in \mathbb{R}$ so that $k_1 = k(j_1) = 1$ and $k_m = k(j_m) = k_{\mathrm{m}}$ under

$$k(j_i) = 1.01 - \frac{1}{1+e^{j_i}}.$$

The remaining $j_i, i = 2, \ldots, (m-1)$ are spaced evenly between j_1 and j_m to fill out the ladder $k_i = k(j_i), i = 1, \ldots, (m-1)$.

By way of comparison, consider generating the three different types of ladder with identical minimum inverse temperature $k_{\mathrm{m}} = 0.1$, the default setting in `tgp`.

```
> geo <- default.itemps(type="geometric")
> har <- default.itemps(type="harmonic")
> sig <- default.itemps(type="sigmoidal")
```

The plots in Figure 15 show the resulting inverse temperature ladders, and their logarithms. Observe how, relative to the geometric ladder, the harmonic ladder has a higher concentration

```
> par(mfrow=c(2,1))
> all <- cbind(geo$k, har$k, sig$k)
> matplot(all, pch=21:23,
+         main="inv-temp ladders", xlab="indx", ylab="itemp")
> legend("topright", pch=21:23,
+        c("geometric","harmonic","sigmoidal"), col=1:3)
> matplot(log(all), pch=21:23,
+         main="log(inv-temp) ladders", xlab="indx", ylab="itemp")
```

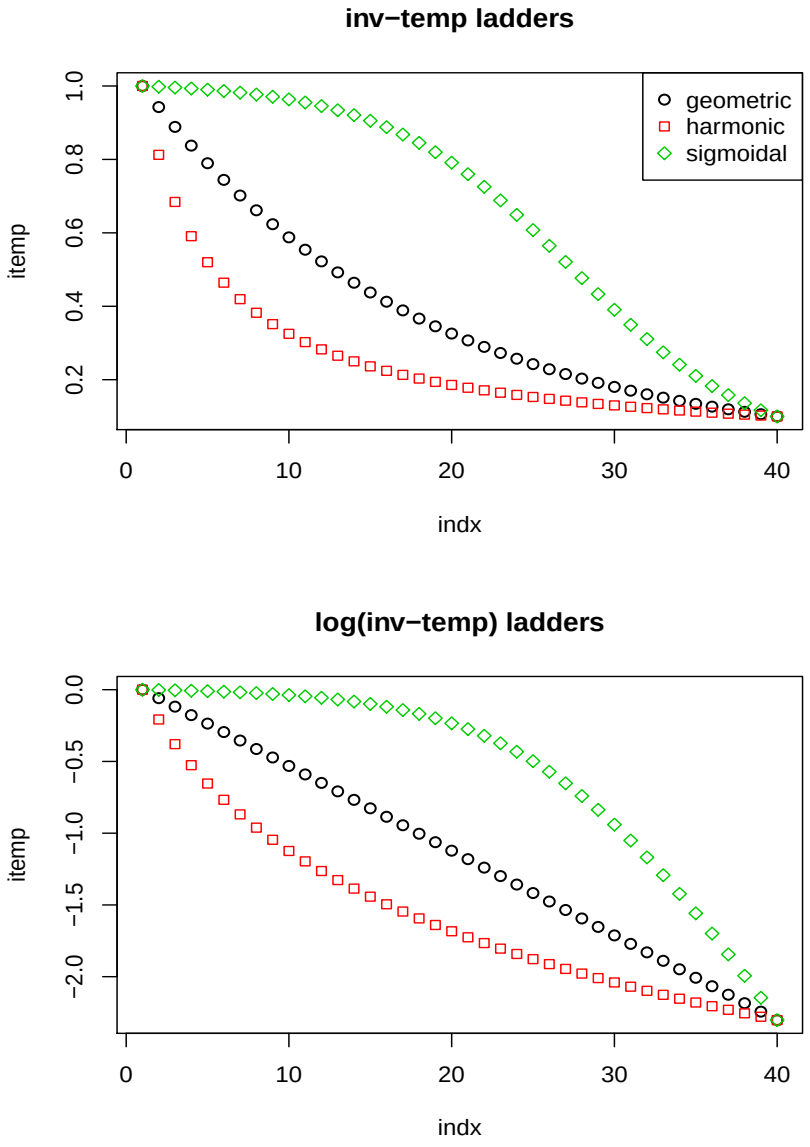

Figure 15: Three different inverse temperature ladders, each with $m = 40$ temperatures starting at $k_1 = 1$ and ending at $k_m = 0.1$

of inverse temperatures near zero, whereas the sigmoidal ladder has a higher concentration near one.

Once a suitable ladder has been chosen, the `tgp` package implementation of ST follows the suggestions of Geyer & Thompson [9] in setting the pseudo–prior, starting from a uniform p_0. First, p_0 is adjusted by *stochastic approximation*: add $c_0/[m(t+n_0)]$ to $\log p_0(k)$ for each $k_i \neq k^{(t)}$ and subtract $c_0/(t+n_0)$ from $\log p_0(k^{(t)})$ over $t = 1, \ldots, B$ *burn–in* MCMC rounds sampling from the joint posterior of (θ, k). Then, p_0 is normalized to obtain p_1. Before subsequent runs, specified via an `R >= 2` argument, *occupation numbers* $o(k_i) = \sum_{t=1}^{B} 1_{\{k^{(t)}=k_i\}}$, are used update $p(k_i) \propto p_1(k_i)/o(k_i)$. Note that, in this setting, the `R` argument is used to update the pseudo–prior only, not to restart the Markov chain.

4.2 Importance sampling from tempered distributions

ST provides us with $\{(\theta^{(t)}, k^{(t)}) : t = 1, \ldots, T\}$, where $\theta^{(t)}$ is an observation from $\pi_{k^{(t)}}$. It is convenient to write $\mathcal{T}_i = \{t : k^{(t)} = k_i\}$ for the index set of observations at the i^{th} temperature, and let $T_i = |\mathcal{T}_i|$. Let the vector of observations at the i^{th} temperature collect in $\boldsymbol{\theta}_i = (\theta_{i1}, \ldots, \theta_{iT_i})$, so that $\{\theta_{ij}\}_{j=1}^{T_i} \sim \pi_{k_i}$. Each vector $\boldsymbol{\theta}_i$ can be used to construct an IS estimator of $E_\pi\{h(\theta)\}$ by setting

$$\hat{h}_i = \frac{\sum_{j=1}^{T_i} w_i(\theta_{ij}) h(\theta_{ij})}{\sum_{j=1}^{T_i} w_i(\theta_{ij})} \equiv \frac{\sum_{j=1}^{T_i} w_{ij} h(\theta_{ij})}{W_i},$$

say. That is, rather than obtain one estimator from ST (at the cold temperature), we can obtain m estimators (one at each temperature) via IS. The efficiency of each estimator, $i = 1, \ldots, m$ can be measured through its variance, but unfortunately this can be difficult to calculate in general. As a result, the notion of *effective sample size* [18] (ESS) plays an important role in the study of IS estimators. Denote the vector of IS weights at the i^{th} temperature as $\mathbf{w}_i = \mathbf{w}_i(\boldsymbol{\theta}_i) = (w_i(\theta_{i1}), \ldots, w_i(\theta_{iT_i}))$, where $w_i(\theta) = \pi(\theta)/\pi_{k_i}(\theta)$. The ESS of $\hat{h}_i$ is defined by

$$\text{ESS}(\mathbf{w}_i) = \frac{T}{1 + \text{cv}^2(\mathbf{w}_i)}, \tag{8}$$

where $\text{cv}(\mathbf{w}_i)$ is the *coefficient of variation* of the weights (in the i^{th} temperature), given by

$$\text{cv}^2(\mathbf{w}_i) = \frac{\sum_{t=1}^{T} (w(\theta^{(t)}) - \bar{w})^2}{(T-1)\bar{w}^2}, \qquad \text{where} \qquad \bar{w} = T^{-1} \sum_{t=1}^{T} w(\theta^{(t)}).$$

In R:

```
> ESS <- function(w)
+ {
+   mw <- mean(w)
+   cv2 <- sum((w-mw)^2)/((length(w)-1)*mw^2)
+   ess <- length(w)/(1+cv2)
```

```
+   return(ess)
+ }
```

This should not be confused with the concept of *effective sample size due to autocorrelation* [17] (due to serially correlated samples coming from a Markov chain as in MCMC) as implemented by the `effectiveSize` function in the `coda` package [26] for R.

Before attempting to combine m IS estimators it is fruitful backtrack briefly to obtain some perspective on the topic of applying IS with a *single* tempered proposal distribution. Jennison [15] put this idea forward more than a decade ago, although the question of how to choose the best temperature was neither posed or resolved. It is clear that larger k leads to lower variance estimators (and larger ESS), but at the expense of poorer mixing in the Markov chain. It can be shown that the optimal inverse temperature k^* for IS, in the sense of constructing a minimum variance estimator, may be significantly lower than one [12]. However, the variance of such an estimator will indeed become unbounded as $k \to 0$, just as ESS $\to 0$. Needless to say, the choice of how to best pick the best temperatures (for ST or IS) is still an open problem. But in the context of the family of tempered distributions used by ST for mixing considerations, this means that the discarded samples obtained when $k^{(t)} < 1$ may actually lead to more efficient estimators than the ones saved from the cold distribution. So ST is wastefull indeed.

However, when combining IS estimators from the multiple temperatures used in ST, the deleterious effect of the high variance ones obtained at high temperature must be mitigated. The possible strategies involved in developing such a meta-estimator comprise the *importance tempering* (IT) family of methods. The idea is that small ESS will indicate high variance IS estimators which should be relegated to having only a small influence on the overall estimator.

4.3 An optimal way to combine IS estimators

It is natural to consider an overall meta-estimator of $E_\pi\{h(\theta)\}$ defined by a convex combination:

$$\hat{h}_\lambda = \sum_{i=1}^{m} \lambda_i \hat{h}_i, \qquad \text{where} \qquad 0 \leq \lambda_i \leq \sum_{i=1}^{m} \lambda_i = 1. \tag{9}$$

Unfortunately, if $\lambda_1, \ldots, \lambda_m$ are not chosen carefully, $\mathrm{Var}(\hat{h}_\lambda)$, can be nearly as large as the largest $\mathrm{Var}(\hat{h}_i)$ [25], due to the considerations alluded to in Section 4.2. Notice that ST is recovered as a special case when $\lambda_1 = 1$ and $\lambda_2, \ldots, \lambda_m = 0$. It may be tempting to choose $\lambda_i = W_i/W$, where $W = \sum_{i=1}^{m} W_i$. The resulting estimator is equivalent to

$$\hat{h} = W^{-1} \sum_{t=1}^{T} w(\theta^{(t)}, k^{(t)}) h(\theta^{(t)}), \qquad \text{where} \qquad W = \sum_{t=1}^{T} w(\theta^{(t)}, k^{(t)}), \tag{10}$$

and $w(\theta, k) = \pi(\theta)/\pi(\theta)^k = \pi(\theta)^{1-k}$. It can lead to a very poor estimator, even compared to ST, as will be demonstrated empirically in the examples to follow shortly.

Observe that we can equivalently write

$$\hat{h}_\lambda = \sum_{i=1}^{m} \sum_{j=1}^{T_i} w_{ij}^\lambda h(\theta_{ij}), \qquad \text{where} \qquad w_{ij}^\lambda = \lambda_i w_{ij} / W_i. \tag{11}$$

Let $\mathbf{w}^\lambda = (w_{11}^\lambda, \ldots, w_{1T_1}^\lambda, w_{21}^\lambda, \ldots, w_{2T_2}^\lambda, \ldots, w_{m1}^\lambda, \ldots, w_{mT_m}^\lambda)$. Attempting to choose $\lambda_1, \ldots, \lambda_m$ to minimize $\mathrm{Var}(\hat{h}_\lambda)$ directly can be difficult. Moreover, for the applications that we have in mind, it is important that our estimator can be constructed without knowledge of the normalizing constants of $\pi_{k_1}, \ldots, \pi_{k_m}$, and without evaluating the MH transition kernels $\mathcal{K}_{\pi_{k_i}}(\cdot, \cdot)$. It is for this reason that methods like the *balance heuristic* [37], MCV [25], or population Monte Carlo (PMC) [5] cannot be applied. Instead, we seek maximize the effective sample size of $\hat{h}_\lambda$ in (9), and look for an $O(T)$ operation to determine the optimal λ^*.

Among estimators of the form (9), it can be shown [12] that $\mathrm{ESS}(\mathbf{w}^\lambda)$ is maximized by $\lambda = \lambda^*$, where, for $i = 1, \ldots, m$,

$$\lambda_i^* = \frac{\ell_i}{\sum_{i=1}^m \ell_i}, \qquad \text{and} \qquad \ell_i = \frac{W_i^2}{\sum_{j=1}^{T_i} w_{ij}^2}.$$

The efficiency of each IS estimator $\hat{h}_i$ can be measured through $\mathrm{ESS}(\mathbf{w}_i)$. Intuitively, we hope that with a good choice of λ, the ESS (8) of $\hat{h}_\lambda$, would be close to the sum over i of the effective sample sizes each of $\hat{h}_i$. This is indeed the case for $\hat{h}_{\lambda^*}$, because it can be shown [12] that

$$\mathrm{ESS}(\mathbf{w}^{\lambda^*}) \geq \sum_{i=1}^{m} \mathrm{ESS}(\mathbf{w}_i) - \frac{1}{4} - \frac{1}{T}.$$

In practice we have found that this bound is conservative and that in fact $\mathrm{ESS}(\mathbf{w}^{\lambda^*}) \geq \sum_{i=1}^m \mathrm{ESS}(\mathbf{w}_i)$, as will be shown empirically in the examples that follow. Thus our optimally–combined IS estimator has a highly desirable and intuitive property in terms of its effective sample size: that the whole is greater than the sum of its parts.

$\mathrm{ESS}(\mathbf{w}^{\lambda^*})$ depends on $\mathrm{ESS}(\mathbf{w}_i)$ which in turn depend on the k_i. Smaller k_i will lead to better mixing in the Markov chain, but lower $\mathrm{ESS}(\mathbf{w}_i)$. Therefore, we can expect that the geometric and sigmoidal ladders will fare better than the harmonic ones, so long as the desired improvements in mixing are achieved. In the examples to follow, we shall see that the sigmoidal ladder does indeed leader to higher $\mathrm{ESS}(\mathbf{w}^{\lambda^*})$.

4.4 Examples

Here the IT method is shown in action for `tgp` models. IT is controlled in `b*` functions via the `itemps` argument: a `data.frame` coinciding with the output of the `default.itemps` function. The `lambda` argument to `default.itemps` can be used to base posterior predictive inference the other IT heuristics: ST and the naïve approach (10). Whenever the argument `m = 1` is used with `k.min != 1` the resulting estimator is constructed via tempered importance sampling at the single inverse temperature `k.min`, in the style of Jennison [15] as outlined

in Section 4.2. The parameters c_0 and n_0 for stochastic approximation of the pseudo–prior can be specified as a 2–vector `c0n0` argument to `default.itemps`. In the examples which follow we simply use the default configuration of the IT method, adjusting only the minimum inverse temperature via the `k.min` argument.

Before delving into more involved examples, we illustrate the stages involved in a small run of importance tempering (IT) on the exponential data from Section 3.3 of [11]. The data can be obtained as:

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

Now, consider applying IT to the Bayesian treed LM with a small geometric ladder. A warning will be given if the default setting of `bprior="bflat"` is used, as this (numerically) improper prior can lead to improper posterior inference at high temperatures.

```
> its <- default.itemps(m=10)
> exp.btlm <- btlm(X=X,Z=Z, bprior="b0", R=2, itemps=its, pred.n=FALSE,
+                  BTE=c(1000,3000,2))

burn in: [with stoch approx (c0,n0)=(100,1000)]
**GROW** @depth 0: [1,0.25], n=(37,43)
**GROW** @depth 1: [1,0.1], n=(19,12)
**PRUNE** @depth 1: [1,0.1]
**GROW** @depth 1: [2,0.1], n=(12,37)
**PRUNE** @depth 1: [2,0.1]
**PRUNE** @depth 0: [1,0.2]
**GROW** @depth 0: [2,0.05], n=(12,68)
**PRUNE** @depth 0: [2,0.1]
**GROW** @depth 0: [1,0.45], n=(58,22)
**GROW** @depth 1: [2,0.5], n=(47,11)
**PRUNE** @depth 1: [2,0.5]
**GROW** @depth 1: [2,0.5], n=(50,12)
**PRUNE** @depth 1: [2,0.5]
r=1000 d=[0] [0]; n=(62,18) k=0.16681
**GROW** @depth 1: [2,0.5], n=(50,12)
r=2000 d=[0] [0] [0]; n=(48,14,18) k=1
**PRUNE** @depth 1: [1,0.5]
**GROW** @depth 1: [1,0.5], n=(54,13)
r=3000 d=[0] [0] [0]; n=(54,13,13) k=0.1

Sampling @ nn=0 pred locs:
r=1000 d=[0] [0] [0]; mh=3 n=(51,12,17) k=0.599484
r=2000 d=[0] [0] [0]; mh=3 n=(51,11,18) k=0.278256
```

```
Grow: 3.629%, Prune: 3.226%, Change: 44.21%, Swap: 33.04%
finished repetition 1 of 2

burn in:
**GROW** @depth 2: [2,0.1], n=(15,33)
**PRUNE** @depth 2: [2,0.1]
r=1000 d=[0] [0] [0]; mh=3 n=(48,14,18) k=1

Sampling @ nn=0 pred locs:
**CPRUNE** @depth 0: var=2, val=0.55->0.5, n=(60,20)
**GROW** @depth 1: [1,0.4], n=(41,16)
r=1000 d=[0] [0] [0]; mh=3 n=(45,12,23) k=0.278256
**PRUNE** @depth 1: [1,0.45]
**GROW** @depth 1: [1,0.45], n=(45,12)
r=2000 d=[0] [0] [0]; mh=3 n=(45,12,23) k=0.599484
Grow: 2.993%, Prune: 2.368%, Change: 35.56%, Swap: 23.45%
finished repetition 2 of 2

effective sample sizes:
0: itemp=1, len=96, ess=96
1: itemp=0.774264, len=168, ess=18.6091
2: itemp=0.599484, len=394, ess=6.44361
3: itemp=0.464159, len=92, ess=1.3093
4: itemp=0.359381, len=268, ess=1.06636
5: itemp=0.278256, len=458, ess=1.01374
6: itemp=0.215443, len=200, ess=2.66161
7: itemp=0.16681, len=152, ess=1.01377
8: itemp=0.129155, len=76, ess=0.987013
9: itemp=0.1, len=96, ess=0.989691
total: len=2000, ess.sum=130.094, ess(w)=130.212
lambda-combined ess=130.212
```

Notice how the MCMC inference procedure starts with $B + T = 4000$ rounds of stochastic approximation (initial adjustment of the pseudo–prior) in place of typical (default) the $B = 1000$ burn–in rounds. Then, the first round of sampling from the posterior commences, over $T = 2000$ rounds, during which the observation counts in each temperature are tallied. The progress meter shows the current temperature the chain is in, say `k=0.629961`, after each of 1000 sampling rounds. The first repeat starts with a pseudo–prior that has been adjusted by the observation counts, which continue to be accumulated throughout the entire procedure (i.e., they are never reset). Any subsequent repeats begin after a similar (re-)adjustment.

Before finishing, the routine summarizes the sample size and effective sample sizes in each rung of the temperature ladder. The number of samples is given by `len`, and the ESS by `ess`. These quantities can also be recovered via `traces`, as shown later. The ESS of

the optimal combined IT sample is the last quantity printed. This, along with the ESS and total numbers of samples in each temperature, can also be obtained via the tgp-class output object.

```
> exp.btlm$ess

$combined
[1] 130.212

$each
           k count        ess
1  1.0000000    96 96.0000000
2  0.7742637   168 18.6090982
3  0.5994843   394  6.4436091
4  0.4641589    92  1.3093041
5  0.3593814   268  1.0663596
6  0.2782559   458  1.0137435
7  0.2154435   200  2.6616119
8  0.1668101   152  1.0137716
9  0.1291550    76  0.9870130
10 0.1000000    96  0.9896907
```

4.4.1 Motorcycle accident data

Recall the motorcycle accident data of Section 3.4 of the first tgp vignette [11]. Consider using IT to sample from the posterior distribution of the treed GP LLM model using the geometric temperature ladder.

```
> library(MASS)
> moto.it <- btgpllm(X=mcycle[,1], Z=mcycle[,2], BTE=c(2000,52000,10),
+          bprior="b0", R=3, itemps=geo, trace=TRUE, pred.n=FALSE, verb=0)
```

Out of a total of 15600 samples from the joint chain, the resulting (optimally combined) ESS was:

```
> moto.it$ess$combined

[1] 1090.111
```

Alternatively, $\mathbf{w}^{\lambda^*}$ can be extracted from the traces, and used to make the ESS calculation directly.

```
> p <- moto.it$trace$post
> ESS(p$wlambda)

[1] 1090.111
```

The unadjusted weights **w** are also available from `trace`. We can see that the naïve choice of $\lambda_i = W_i/W$, leading to the estimator in (10), has a clearly inferior effective sample size.

```
> ESS(p$w)

[1] 11.16957
```

To see the benefit of IT over ST we can simply count the number of samples obtained when $k^{(t)} = 1$. This can be accomplished in several ways: either via the traces or through the output object.

```
> as.numeric(c(sum(p$itemp == 1), moto.it$ess$each[1,2:3]))

[1] 393 393 393
```

That is, (optimal) IT gives effectively 2.77 times more samples. The naïve combination, leading to the estimator in (10), yields an estimator with an effective sample size that is 3% of the number of samples obtained under ST.

Now, we should like to compare to the MCMC samples obtained under the same model, without IT.

```
> moto.reg <- btgpllm(X=mcycle[,1], Z=mcycle[,2], BTE=c(2000,52000,10),
+          R=3, bprior="b0", trace=TRUE, pred.n=FALSE, verb=0)
```

The easiest comparison to make is to look at the heights explored under the three chains: the regular one, the chain of heights visited at all temperatures (combined), and those obtained after applying IT via re-weighting under the optimal combination λ^*.

```
> L <- length(p$height)
> hw <- suppressWarnings(sample(p$height, L, prob=p$wlambda, replace=TRUE))
> b <- hist2bar(cbind(moto.reg$trace$post$height, p$height, hw))
```

Figure 16 shows barplots indicating the count of the number of times the Markov chains were in trees of various heights after burn–in. Notice how the tempered chain (denoted "All Temps" in the figure) frequently visits trees of height one, whereas the non–tempered chain (denoted "reg MCMC") never does. The result is that the non–tempered chain underestimates the probability of height two trees and produces a corresponding overestimate of height four trees—which are clearly not supported by the data—even visiting trees of height five. The IT estimator appropriately down–weights height one trees and provides correspondingly more realistic estimates of the probability of height two and four trees.

Whenever introducing another parameter into the model, like the inverse temperature k, it is important to check that the marginal posterior chain for that parameter is mixing well. For ST it is crucial that the chain makes rapid excursions between the cold temperature, the hottest temperatures, and visits each temperature roughly the same number of times.

Figure 17 shows a trace of the posterior samples for k in the motorcycle experiment. Arguably, the mixing in k–space leaves something to be desired. Since it can be very difficult

```
> barplot(b, beside=TRUE, col=1:3, xlab="tree height", ylab="counts",
+         main="tree heights encountered")
> legend("topright", c("reg MCMC", "All Temps", "IT"), fill=1:3)
```

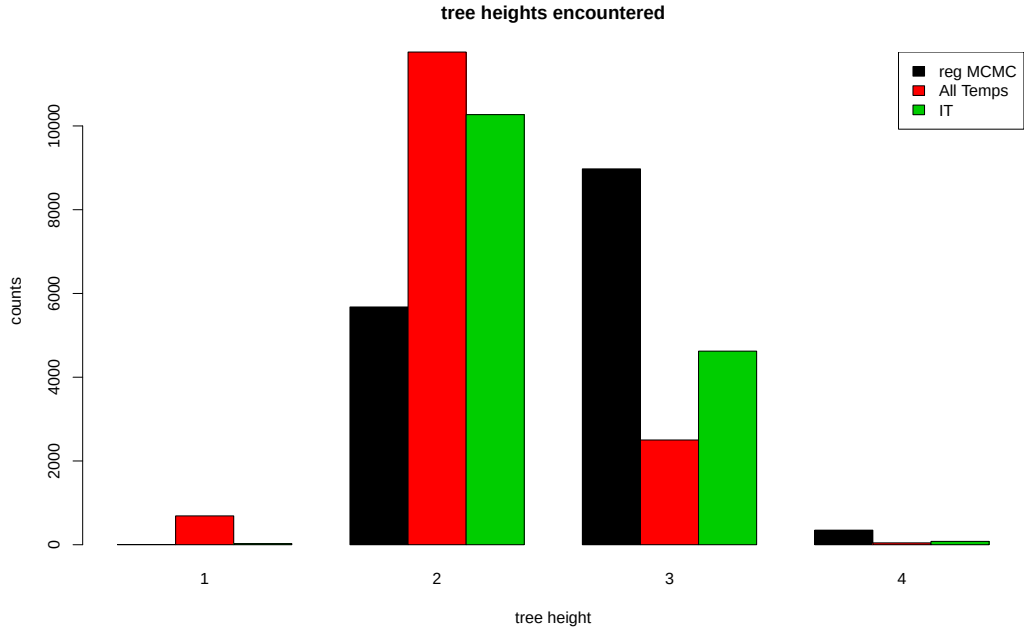

Figure 16: Barplots indicating the counts of the number of times the Markov chains (for regular MCMC, combining all temperatures in the inverse temperature ladder, and those re-weighted via IT) were in trees of various heights for the motorcycle data.

to tune the pseudo–prior and MH proposal mechanism to get good mixing in k–space, it is fortunate that the IT methodology does not rely on the same mixing properties as ST does. Since samples can be obtained from the posterior distribution of the parameters of interest by re-weighting samples obtained when $k < 1$ it is only important that the chain frequently visit low temperatures to obtain good sampling, and high temperatures to obtain good mixing. The actual time spent in specific temperatures, i.e., $k = 1$ is less important. Figure 18 shows the histogram of the inverse temperatures visited in the Markov chain for the motorcycle experiment. Also plotted is a histogram of the *observation counts* in each temperature. The two histograms should have similar shape but different totals. Observation counts are tallied during every MCMC sample after burn–in, whereas the posterior samples of k are thinned (at a rate specified in `BTE[3]`). When the default `trace=FALSE` argument is used only the observation counts will be available in the `tgp`–class object, and these can be used as a surrogate for a trace of k.

The compromise IT approach obtained using the sigmoidal ladder can yield an increase in ESS.

```
> moto.it.sig <- btgpllm(X=mcycle[,1], Z=mcycle[,2], BTE=c(2000,52000,10),
+                        R=3, bprior="b0", krige=FALSE, itemps=sig, verb=0)
```

```
> plot(log(moto.it$trace$post$itemp), type="l", ylab="log(k)", xlab="samples",
+      main="trace of log(k)")
```

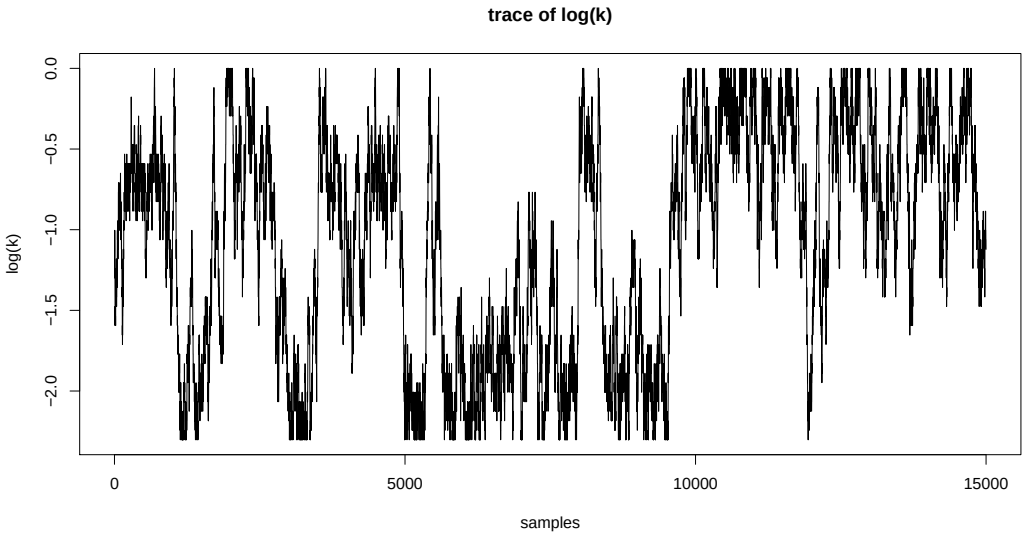

Figure 17: A trace of the MCMC samples from the marginal posterior distribution of the inverse temperature parameter, k, in the motorcycle experiment

Compare the resulting ESS to the one given for the geometric ladder above.

```
> moto.it.sig$ess$combined
```

```
[1] 6530.414
```

Plots of the resulting predictive surface is shown in Figure 19 for comparison with those in Section 1.1 of the first `tgp` vignette [11]. In particular, observe that the transition from the middle region to the right one is much less stark in this tempered version than than in the original—which very likely spent a disproportionate amount of time stuck in a posterior mode with trees of depth three or greater.

4.4.2 Synthetic 2–d Exponential Data

Recall the synthetic 2–d exponential data of Section 3.4 of the tgp vignette [11], where the true response is given by

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

Here, we will take $\mathbf{x} \in [-6, 6] \times [-6, 6]$ with a D–optimal design

```
> Xcand <- lhs(10000, rbind(c(-6,6),c(-6,6)))
> X <- dopt.gp(400, X=NULL, Xcand)$XX
> Z <- exp2d.Z(X)$Z
```

Consider a treed GP LLM model fit to this data using the standard MCMC.

```
> b <- itemps.barplot(moto.it, plot.it=FALSE)
> barplot(t(cbind(moto.it$itemps$counts, b)), col=1:2,
+         beside=TRUE, ylab="counts", xlab="itemps",
+         main="inv-temp observation counts")
> legend("topleft", c("observation counts", "posterior samples"), fill=1:2)
```

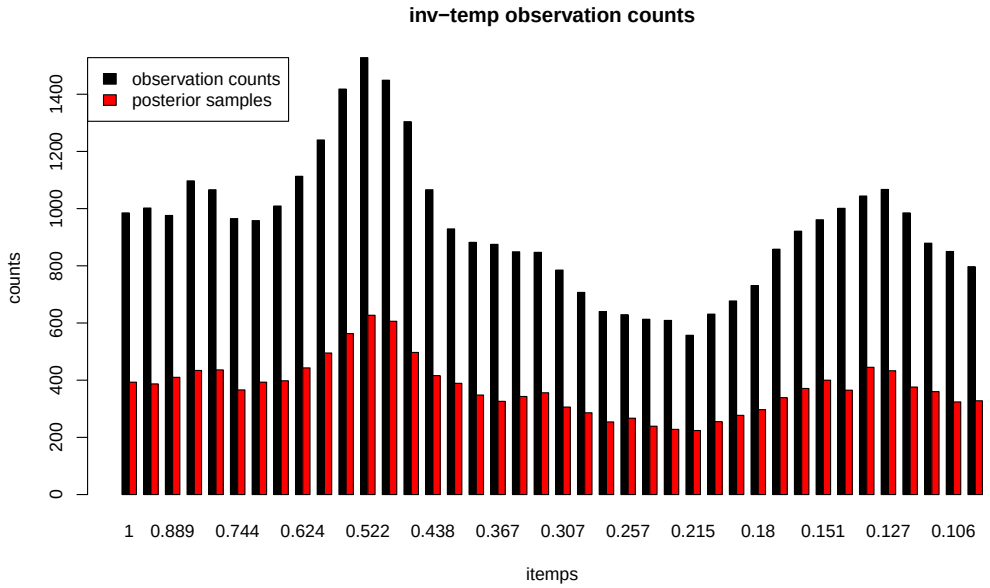

Figure 18: Comparing (thinned) samples from the posterior distribution for the inverse temperature parameter, k, (posterior samples), to the observation counts used to update the pseudo–prior, in the motorcycle experiment

```
> plot(moto.it.sig)
```

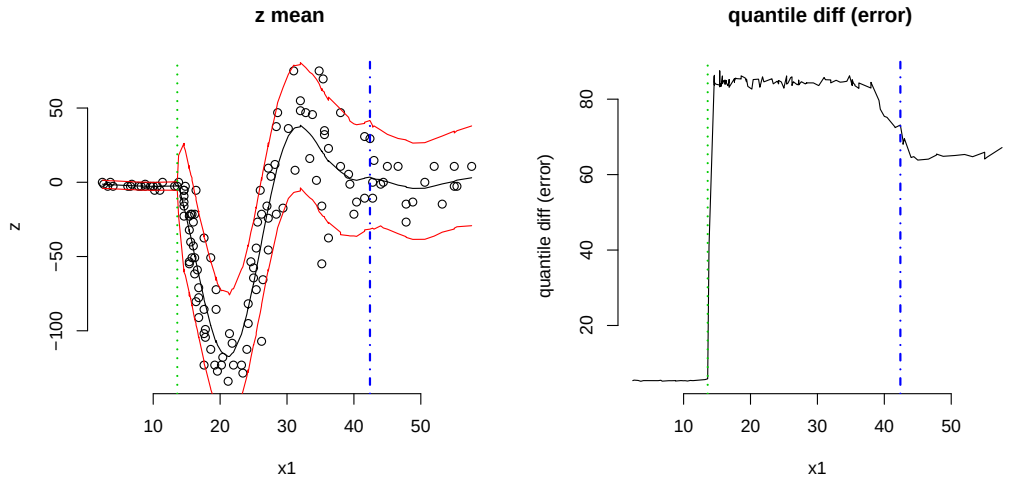

Figure 19: Posterior predictive surface for the motorcycle data, with 90% quantile errorbars, obtained under IT with the sigmoidal ladder.

```
> exp.reg <- btgpllm(X=X, Z=Z, BTE=c(2000,52000,10), bprior="b0",
+                    trace=TRUE, krige=FALSE, R=10, verb=0)

> plot(exp.reg)
```

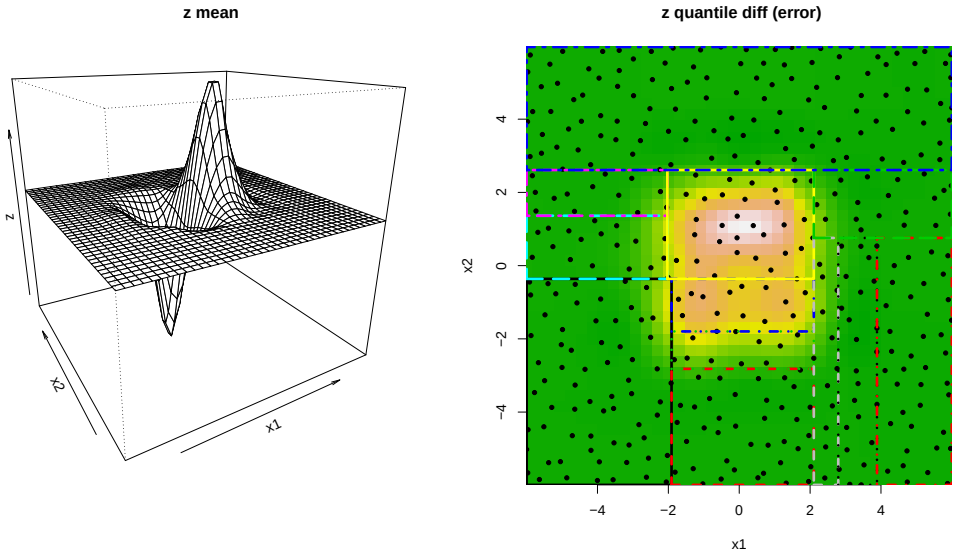

Figure 20: Posterior predictive surface for the 2–d exponential data: mean surface *(left)* and 90% quantile difference *(right)*

Figure 20 shows the resulting posterior predictive surface. The maximum *a' posteriori* (MAP) tree is drawn over the error surface in the *right–hand* plot. The height of this tree can be obtained from the `tgp`-class object.

```
> h <- exp.reg$post$height[which.max(exp.reg$posts$lpost)]
> h

[1] 7
```

It is easy to see that many fewer partitions are actually necessary to separate the interesting, central, region from the surrounding flat region. Figure 21 shows a diagrammatic representation of the MAP tree. Given the apparent over–partitioning in this height 7 tree it would be surprising to find much posterior support for trees of greater height. One might indeed suspect that there are trees with fewer partitions which would have higher posterior probability, and thus guess that the Markov chain for the trees plotted in these figures possibly became stuck in a local mode of tree space while on an excursion into deeper trees.

Now consider using IT. It will be important in this case to have a k_{m} small enough to ensure that the tree occasionally prunes back to the root. We shall therefore use a smaller k_{m}. Generally speaking, some pilot tuning may be necessary to choose an appropriate k_{m} and number of rungs m, although the defaults should give adequate performance in most cases.

```
> tgp.trees(exp.reg, "map")
```

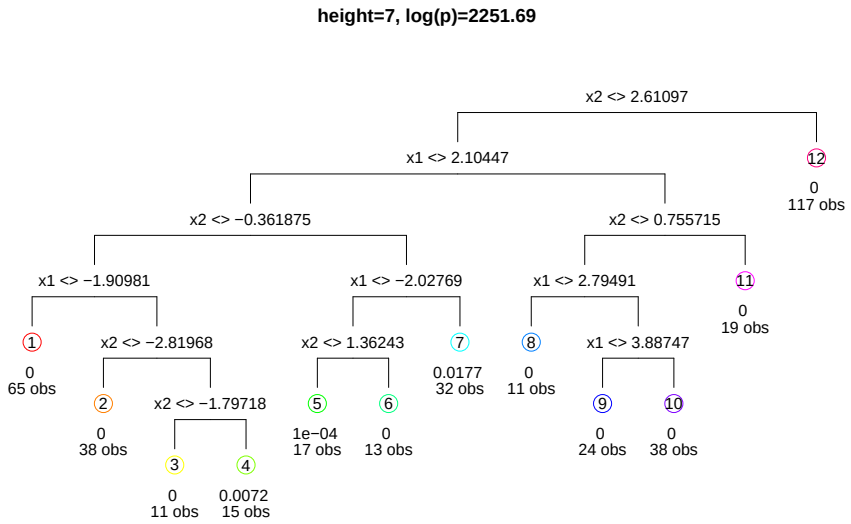

Figure 21: Diagrammatic depiction of the maximum *a' posteriori* (MAP) tree for the 2–d exponential data under standard MCMC sampling

```
> its <- default.itemps(k.min=0.02)
> exp.it <- btgpllm(X=X, Z=Z, BTE=c(2000,52000,10), bprior="b0",
+                   trace=TRUE, krige=FALSE, itemps=its, R=10, verb=0)
```

As expected, the tempered chain moves more rapidly throughout tree space by accepting more tree proposals. The acceptance rates of tree operations can be accessed from the `tgp`–class object.

```
> exp.it$gpcs

        grow     prune  change      swap
1 0.07231571 0.0725638 0.81346 0.5548426

> exp.reg$gpcs

        grow       prune    change     swap
1 0.01099707 0.008306095 0.6560099 0.347478
```

The increased rate of *prune* operations explains how the tempered distributions helped the chain escape the local modes of deep trees.

We can quickly compare the effective sample sizes of the three possible estimators: ST, naïve IT, and optimal IT.

```
> p <- exp.it$trace$post
> data.frame(ST=sum(p$itemp == 1), nIT=ESS(p$w), oIT=exp.it$ess$combined)
```

```
    ST      nIT      oIT
1 1955 128.7141 2402.957
```

Due to the thinning in the Markov chain (`BTE[3] = 10`) and the traversal between $m = 10$ temperatures in the ladder, we can be reasonably certain that the 2403 samples obtained via IT from the total of 50000 samples obtained from the posterior are far less correlated than the ones obtained via standard MCMC.

As with the motorcycle data, we can compare the tree heights visited by the two chains.

```
> L <- length(p$height)
> hw <- suppressWarnings(sample(p$height, L, prob=p$wlambda, replace=TRUE))
> b <- hist2bar(cbind(exp.reg$trace$post$height, p$height, hw))

> barplot(b, beside=TRUE, col=1:3, xlab="tree height", ylab="counts",
+         main="tree heights encountered")
> legend("topright", c("reg MCMC", "All Temps", "IT"), fill=1:3)
```

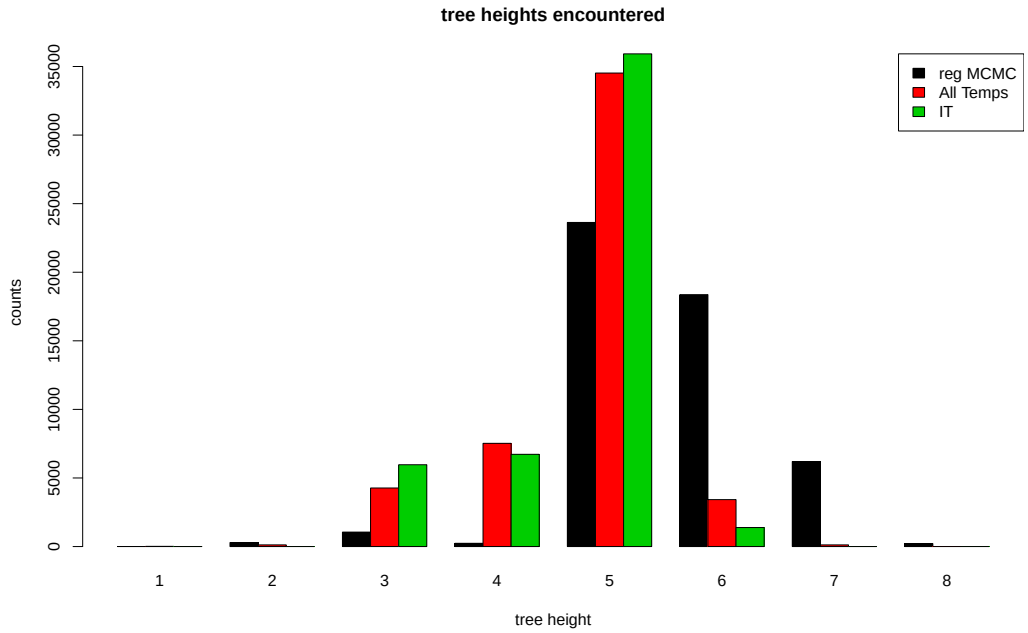

Figure 22: Barplots indicating the counts of the number of times the Markov chains (for regular MCMC, combining all temperatures in the inverse temperature ladder, and those re-weighted via IT) were in trees of various heights for the 2–d exponential data.

Figure 22 shows a barplot of `b`, which illustrates that the tempered chain frequently visited shallow trees. IT with the optimal weights shows that the standard MCMC chain missed many trees of height three and four with considerable posterior support.

To more directly compare the mixing in tree space between the ST and tempered chains, consider the trace plots of the heights of the trees explored by the chains shown in Figure 23. Despite being restarted 10 times, the regular MCMC chain (almost) never visits trees of

```
> ylim <- range(p$height, exp.reg$trace$post$height)
> plot(p$height, type="l", main="trace of tree heights",
+      xlab="t", ylab="height", ylim=ylim)
> lines(exp.reg$trace$post$height, col=2)
> legend("topright", c("tempered", "reg MCMC"), lty=c(1,1), col=1:2)
```

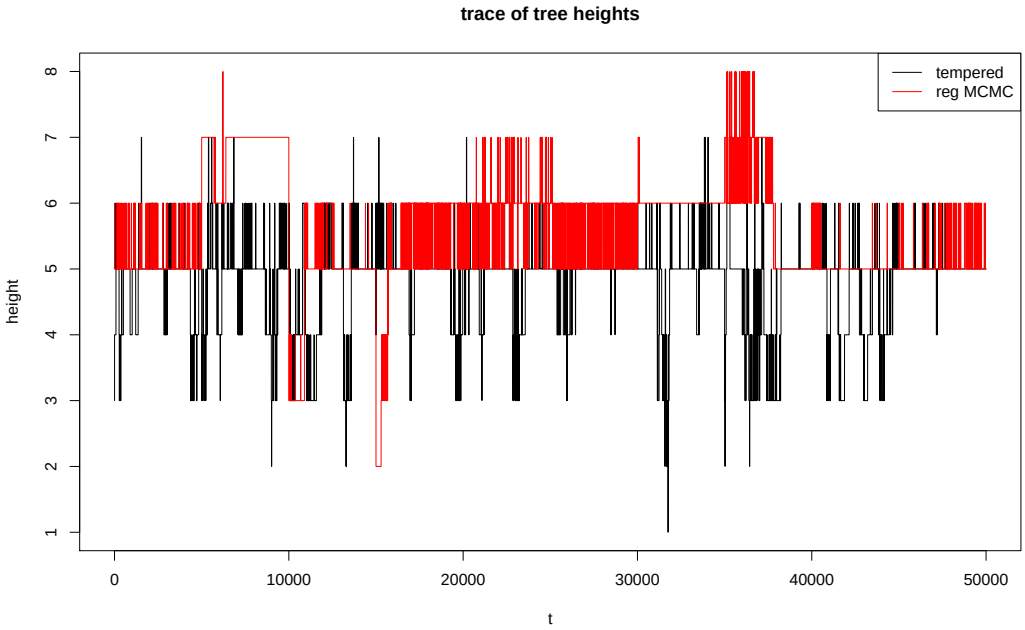

Figure 23: Traces of the tree heights obtained under the two Markov chains (for regular MCMC, combining all temperatures in the inverse temperature ladder) on the 2–d exponential data.

height less than five after burn–in and instead makes rather lengthy excursions into deeper trees, exploring a local mode in the posterior. In contrast, the tempered chain frequently prunes back to the tree root, and consequently discovers posterior modes in tree heights three and four.

To conclude, a plot of the posterior predictive surface is given in Figure 24, where the MAP tree is shown both graphically and diagrammatically.

Acknowledgments

This work was partially supported by research subaward 08008-002-011-000 from the Universities Space Research Association and NASA, NASA/University Affiliated Research Center grant SC 2003028 NAS2-03144, Sandia National Laboratories grant 496420, National Science Foundation grants DMS 0233710 and 0504851, and Engineering and Physical Sciences Research Council Grant EP/D065704/1. The authors would like to thank their Ph.D. advisor, Herbie Lee, whose contributions and guidance in this project have been invaluable throughout. Finally, we would like to thank two anonymous referees whose many helpful

```
> plot(exp.it)
> tgp.trees(exp.it, "map")
```

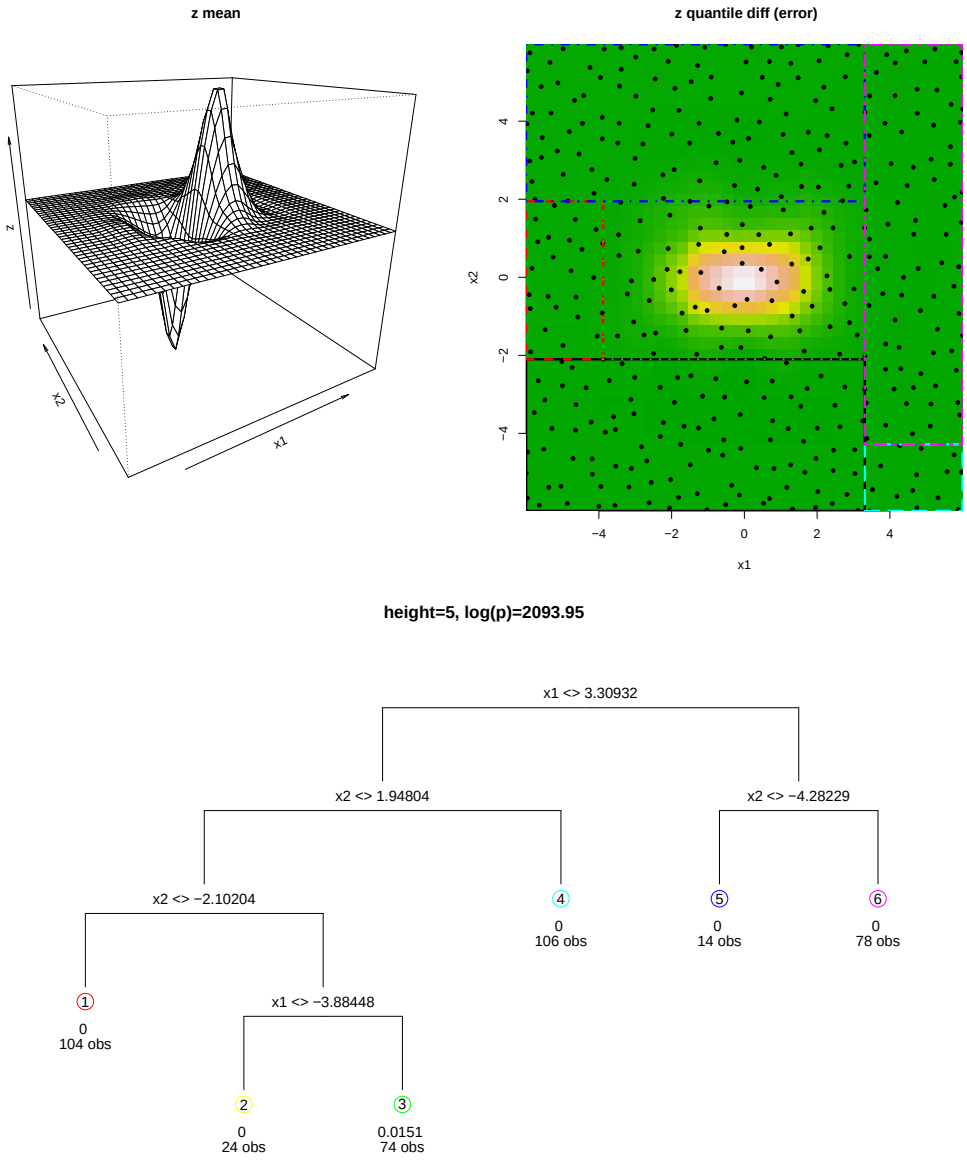

Figure 24: 2–d exponential data fit with IT. *Top:* Posterior predictive mean surface for the 2d–exponential, with the MAP tree overlayed. *Bottom:* diagrammatic representation of the MAP tree.

comments improved the paper.

References

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

[2] 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.

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

[4] S. Da Veiga, F. Wahl, and F. Gamboa. Local polynomial estimation for sensitivity analysis on models with correlated inputs. *Technometrics*, 51:452–463, 2009.

[5] R. Douc, A. Guillin, J.-M. Marin, and C.P. Robert. Minimum variance importance sampling via population monte carlo. Technical report, CEREMADE, Université Paris Dauphine, and CREST, INSEE, Paris, 2007.

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

[7] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.

[8] C.J. Geyer. Markov chain Monte Carlo maximum likelihood. In *Computing Science and Statistics: Proceedings of the 23rd Symposium on the Interface*, pages 156–163, 1991.

[9] C.J. Geyer and E.A. Thompson. Annealing Markov chain Monte Carlo with applications to ancenstral inference. *Journal of the American Statistical Association*, 90:909–920, 1995.

[10] David Ginsbourger, Rodolphe Le Riche, and Laurent Carraro. A multi-points criterion for deterministic parallel global optimization based on Gaussian processes. HAL: hal-00260579, 2009.

[11] Robert B. Gramacy. `tgp`: An `R` package for Bayesian nonstationary, semiparametric nonlinear regression and design by treed gaussian process models. *Journal of Statistical Software*, 19(9), 6 2007.

[12] Robert B. Gramacy, Richard J. Samworth, and Ruth King. Importance tempering. Technical Report 0707.4242, ArXiv, 2009. to appear in Statistics and Computing.

[13] W.K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57:97–109, 1970.

[14] T. Homma and A. Saltelli. Importance measures in global sensitivity analysis of nonlinear models. *Reliability engineering and system safety*, 52:1–17, 1996.

[15] C. Jennison. Discussion on the meeting on the gibbs sampler and other Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society, Series B*, 55:54–56, 1993.

[16] 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.

[17] Robert E. Kass, Bradley P. Carlin, Andrew Gelman, and Radford M. Neal. Markov chain monte carlo in practice: A roundtable discussion. *The American Statistician*, 52(2):93–100, May 1998.

[18] J. S. Liu. *Monte Carlo Strategies in Scientific Computing*. Springer, New York, 2001.

[19] A. Marrel, B. Iooss, B. Laurent, and O. Roustant. Calculations of sobol indices for the gaussian process metamodel. *Reliability Engineering & System Safety*, 94:742–751, 2009.

[20] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and R. Teller. Equations of state calculations by fast computing machine. *Journal of Chemical Physics*, 21:1087–1091, 1953.

[21] R. D. Morris, A. Kottas, M. Taddy, R. Furfaro, and B. Ganapol. A statistical framework for the sensitivity analysis of radiative transfer models. *IEEE Transactions on Geoscience and Remote Sensing*, To appear.

[22] Radford M. Neal. Sampling from multimodal distributions using tempered transition. *Statistics and Computing*, 6:353–366, 1996.

[23] Radford M. Neal. Annealed importance sampling. *Statistics and Computing*, 11:125–129, 2001.

[24] J.E. Oakley and A. O'Hagan. Probabilistic sensitivity analysis of complex models: a Bayesian approach. *Journal of the Royal Statistical Society Series B*, 66:751–769, 2004.

[25] Art Owen and Yi Zhou. Safe and effective importance sampling. *Journal of the American Statstical Association*, 95(449):135–143, March 2000.

[26] Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines. *coda: Output analysis and diagnostics for MCMC*, 2008. R package version 0.13-3.

[27] Z.G. Qian, H. Wu, and C.F.J. Wu. Gaussian process models for computer experiments with qualitative and quantitative factors. *Technometrics*, 50:383–396, 2009.

[28] Rommel G. Regis and Christine A. Shoemaker. Improved strategies for radial basis function methods for global optimization. *J. of Global Optimization*, 37(1):113–135, 2007.

[29] A. Saltelli, K. Chan, and E.M. Scott, editors. *Sensitivity Analysis*. John Wiley and Sons, 2000.

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

[31] A. Saltelli and S. Tarantola. On the relative importance of input factors in mathematical models: safety assessment for nuclear waste disposal. *Journal of the American Statistical Association*, 97:702–709, 2002.

[32] Andrea Saltelli. Making best use of model evaluations to compute sensitivity indices. *Computer Physics Communications*, 145:280–297, 2002.

[33] M. Schonlau, D.R. Jones, and W.J. Welch. Global versus local search in constrained optimization of computer models. In *New Developments and applications in experimental design*, number 34 in IMS Lecture Notes - Monograph Series, pages 11–25. IMS, 1998.

[34] Curtis B. Storlie and Jon C. Helton. Multiple predictor smoothing methods for sensitivity analysis: Description of techniques. *Reliability Engineering & System Safety*, 93:28–54, 2008.

[35] Curtis B. Storlie, Laura P. Swiler, Jon C. Helton, and Cedric J. Sallaberry. Implementation and evaluation of nonparametric regression procedures for sensitivity analysis of computationally demanding models. *Reliability Engineering & System Safety*, 94:1735–1763, 2009.

[36] Matthew A. Taddy, Herbert K. H. Lee, Genetha A. Gray, and Joshua D. Griffin. Bayesian guided pattern search for robust local optimization. *Technometrics*, 51:389–401, 2009.

[37] Eric Veach and Leonidas J. Guibas. Optimally combining sampling techniques for monte carlo rendering. In *SIGGRAPH '95 Conference Proceedings*, pages 419–428, Reading, MA, 1995. Addison–Wesley.

[38] W. J. Welch, R. J. Buck, J. Sacks, H. P. Wynn, T.J Mitchell, and M. D. Morris. Screening, predicting, and computer experiment. *Technometrics*, 34:15–25, 1992.