Scientific Research Publishing

Applied Mathematics, 2020, 11, 613-638
https://www.scirp.org/journal/am
ISSN Online: 2152-7393
ISSN Print: 2152-7385

An Improved Treed Gaussian Process

John Guenther, Herbert K. H. Lee

[1]Department of Statistics and Biostatistics, California State University, Hayward, CA, USA
[2]Department of Statistics, University of California, Santa Cruz, CA, USA
Email: john.guenther@csueastbay.edu

How to cite this paper: Guenther, J. and Lee, H.K.H. (2020) An Improved Treed Gaussian Process. *Applied Mathematics*, **11**, 613-638.
https://doi.org/10.4236/am.2020.117042

Received: June 5, 2020
Accepted: July 19, 2020
Published: July 22, 2020

Abstract

Many black box functions and datasets have regions of different variability. Models such as the Gaussian process may fall short in giving the best representation of these complex functions. One successful approach for modeling this type of nonstationarity is the Treed Gaussian process [1], which extended the Gaussian process by dividing the input space into different regions using a binary tree algorithm. Each region became its own Gaussian process. This iterative inference process formed many different trees and thus, many different Gaussian processes. In the end these were combined to get a posterior predictive distribution at each point. The idea was that when the iterations were combined, smoothing would take place for the surface of the predicted points near tree boundaries. We introduce the Improved Treed Gaussian process, which divides the input space into a single main binary tree where the different tree regions have different variability. The parameters for the Gaussian process for each tree region are then determined. These parameters are then smoothed at the region boundaries. This smoothing leads to a set of parameters for each point in the input space that specify the covariance matrix used to predict the point. The advantage is that the prediction and actual errors are estimated better since the standard deviation and range parameters of each point are related to the variation of the region it is in. Further, smoothing between regions is better since each point prediction uses its parameters over the whole input space. Examples are given in this paper which show these advantages for lower-dimensional problems.

Keywords

Bayesian Statistics, Treed Gaussian Process, Gaussian Process, Emulator, Binary Tree

1. Introduction

A common complication in the analysis of data is changes in variability, either in

the smoothness of the underlying function or as heteroscedasticity. Fully modeling this variability can be computationally prohibitive. The Gaussian process is commonly used as a nonparametric model for complex data, especially in the context of computer simulation modeling or estimation of black box functions, as discussed in [2] [3]. A limitation of the Gaussian process is that it may not represent all regions of the functional domain well when there are regions of different variability. The Treed Gaussian process was developed to handle this situation in a computationally efficient manner [1].

Other examples of non-stationary Gaussian processes include [4]-[11]. Many of these are more computationally intensive than the Treed Gaussian process. [4] uses non-stationary spatial dependence as explained through a constructive "process-convolution" approach, ensuring the resulting covariance structure is valid. [5] introduces a new class of nonstationary covariance functions which allow the model to adapt to spatial surfaces whose variability changes with location. [6] uses a process convolutions-based GP model by convolving a smoothing kernel with a partitioned nonstationarity latent process in the Gaussian process obtained by allowing the variability of the latent process and the kernel size to change across partitions. [7] uses a method which automatically decomposes the spatial domain into disjoint regions within which the process is assumed to be stationary but across regions the data is assumed independent. Uncertainty in the number of disjoint regions, their shapes, and the model within regions is dealt with in a fully Bayesian fashion. [8] uses clustering to determine different regions of variability and a nonstationary Gaussian process across the clusters. [9] uses an approach described as spatially clustered coefficient regression to model variable functions. [10] uses a nonstationary covariance function constructed based on selected partitions to model functions of variability. [11] uses piecewise Gaussian processes to model sharp changes in the covariance matrix due to the change in rock types for geological structures.

The Treed Gaussian process [1], which we compare to our Improved Treed Gaussian process in this paper, models functions with regions of different variability by an iterative inference process which divides up the input space into regions using a binary tree approach to isolate the regions of different variability. Each tree region, for a given iteration, has its own Gaussian process. In the end, the predictions for points in the input space for each iteration are combined to get a posterior predictive distribution for each point. There can be these problems with the Treed Gaussian Process (TGP): 1) The results can vary considerably for different calls to the Treed Gaussian Process (TGP); 2) The smoothing of the different sets of tree iterations near tree boundaries may not emulate the function as well as desired; 3) The errors of the predictions and the error estimates may be larger than desirable in some cases. It does have the advantage that the processing is very efficient with regard to the time it takes for results.

Our Improved Treed Gaussian process takes into account the different regions of variability by dividing the different regions of variability based on a set of gridded training points. This is done using a binary tree algorithm. It differs

from the Treed Gaussian process in that we use a single main tree. The Gaussian process for each tree region is determined providing the process variance and the range parameters. Then the parameters of the different regions are smoothed where they interface each other. This way, each predicted point has its own parameters, whether it lies within a tree region or on the boundary between tree regions. Now each point is predicted based on its own unique covariance matrix. Although the time for processing is increased, there are these advantages: 1) The results can be reproduced with only minor random variation; 2) Smoothing occurs at the region boundaries which emulates the function nicely since the prediction process uses the whole input space; 3) The estimated errors (σ values) are closer to the actual errors. This approach works well in low-dimensional problems, one or two dimensions, although we expect that further research can extend this method to larger dimensions.

The details of this Improved Treed Gaussian emulator are discussed in Section 2. After these details are discussed, Section 3 presents two illustrations of the Improved Treed Gaussian process (herein called the Improved TGP) and compares its results with those of the Treed Gaussian Process (TGP). Lastly, in Section 4, the Improved TGP is applied to real data. Conclusions follow in Section 5.

2. Improved Treed Gaussian Process

The algorithm begins with the evaluation of gridded training points covering the function's input space. The reason for such a grid is that now the point values can be used to get a **variance matrix**. This variance matrix is made up of the variances of each grid point. The variance of each grid point is the variance of the function values for the given point and all neighboring points within the grid distance including those on the diagonals. Next the variance matrix and the gridded training points are displayed in perspective views to give the user an idea of what the binary tree might be. There are four types of functions: 1) Irregular, 2) variable, 3) smooth, and 4) very smooth. **Figure 1** shows an example of each type of function.

The user decides which type of function he/she is emulating. Then the binary tree algorithm is run to determine the binary tree that suits the function best. This algorithm is explained in Subsection 2.2 with details given in Subsection 2.3. After the binary tree is obtained, the individual regions of the binary tree are processed. This is a straightforward application of the Metropolis-Hastings algorithm. For each region, this algorithm determines the process variance, σ^2, the range parameters θ_1 and θ_2 for x_1 and x_2 (or just for x_1 if one dimensional), and the nugget, τ. A prior for the range parameters has been developed for this processing step which will be explained in Subsection 2.4. The parameters for each region are smoothed at their interfaces of other tree regions to obtain a covariance matrix for each predicted point. Smoothing is discussed in Subsection 2.1. The final step is the prediction of points and their errors. This is discussed in Subsection 2.5.

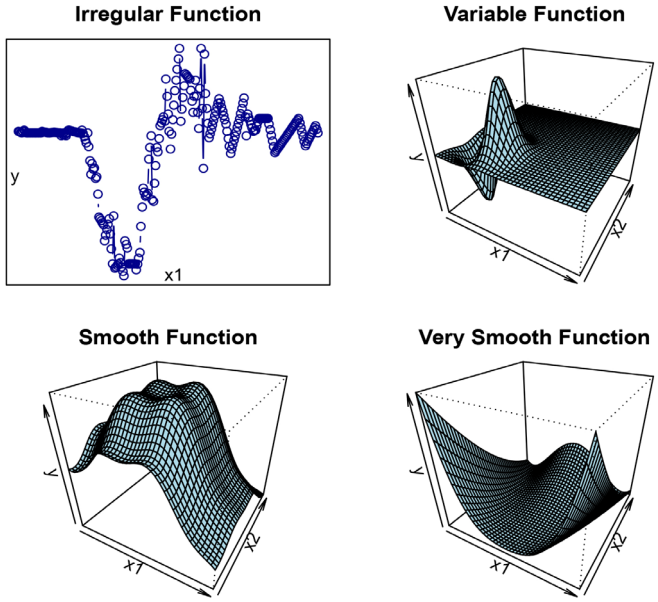

Figure 1. Illustration of four function types: the irregular function is the motorcycle stopping data discussed in Section 4; the variable function is discussed in Section 3.1; the third function is an example of a smooth function; the fourth function is the very smooth Rosenbrock Banana function.

2.1. Smoothing Process Parameters at Region Interfaces

For the Treed Gaussian process [1], smoothing depends on averaging many iterations of different binary trees which center around the regions of different variability of the function. Now we have a single tree and we want smoothing at the interface of specific tree regions. There are two types of smoothing: 1) Smoothing for the process variance; 2) Smoothing for the range parameters. The difference between these two types is due to the fact that we expect a region with a larger process variance should raise the process variance of a smaller process variance region at their interface. The reason for this is that, at the interface of the two regions, errors are likely to be larger than predicted by the smaller process variance. If this smoothing is left out, errors at the boundaries are often underestimated. Figure 2 shows how smoothing takes place at this interface between a region with a larger process variance than the region it interfaces with.

At the interface of two regions with different range values, a smooth transition is made in which both regions are modified at the interface. The region with larger range has its range reduced while the region with smaller range has its range increased. The idea is that the points predicted at the region boundary should share equally in the two ranges. This smoothing allows for the Gaussian process points near the boundary region to better share in the final estimation of points near the boundary. Figure 3 shows this smoothing.

Two dimensional smoothing is more complicated since the smoothed value at a point near a boundary depends on more neighboring points. However, at two region boundaries it turns out to be the same as the one dimensional case.

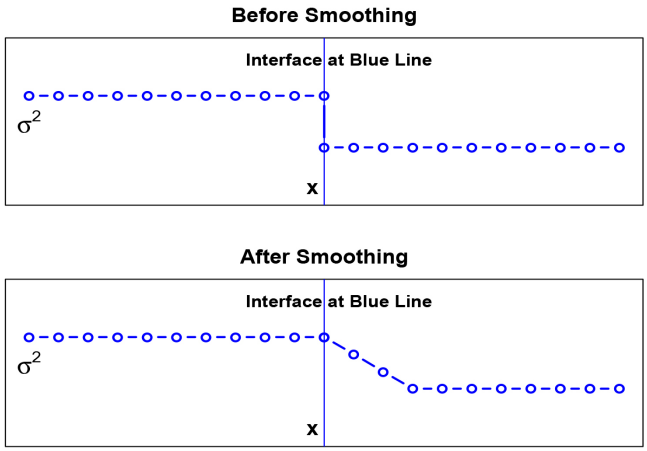

Figure 2. Smoothing for the process variance: the one dimensional view shows how smoothing is done for the interface between a region with a larger process variance and one with a smaller process variance. The larger process variance region modifies the smaller process variance region by raising its variance near the interface.

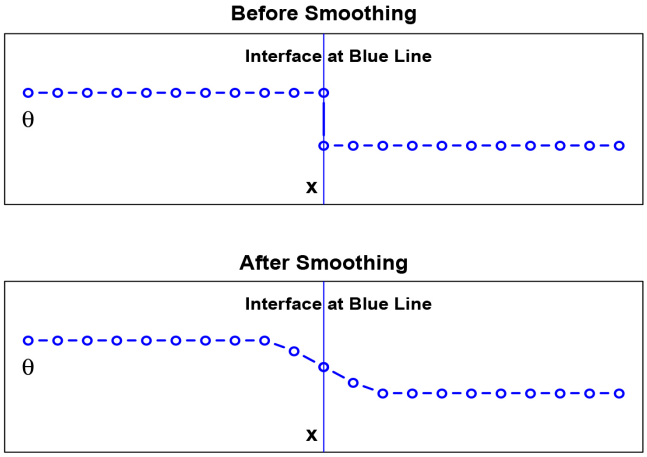

Figure 3. Smoothing for range: the one dimensional view shows how smoothing is done for the interface between a region with a larger range than the one it interfaces with. Both ranges are smoothed at the interface.

2.2. Overview of the Binary Tree Algorithm

The Gaussian process does not model a function very well that has regions of different variability. Many approaches have been taken to circumvent this difficulty [1] [4]-[11]. Our method divides the function input space into different regions using a variance matrix determined by a set of gridded points over the function's input space.

This is illustrated by a simple case where the function varies by only a small amount in one region and varies by a large amount in the other region. The function is shown in **Figure 4** on the left side. The variance matrix is on the right side. There is a significant change in the variance of points moving from

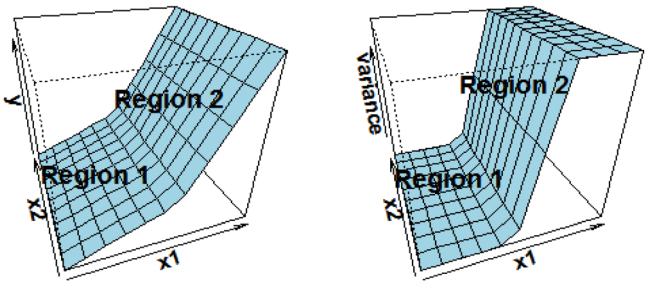

Figure 4. Perspective of simple function and variance matrix: the function is on the left. The variance matrix is on the right. The variance at any grid point is the variance of the function values nearest the point.

left to right along variable x_1. This change divides the function space into two regions as shown in the perspectives. Two Gaussian functions with smoothing at the region boundary are expected to be a better fit to the surface than one Gaussian function. The proposed binary tree algorithm divides the function into two regions of differing variability. The function is:

$$f(x_1, x_2) = \begin{cases} x_1 & \text{if } 0 \le x_1 \le 1 \\ 1 + 2(x_1 - 1) & \text{if } 1 < x_1 \le 1.2 \\ 1.4 + 4(x_1 - 1.2) & \text{if } 1.2 < x_1 \le 2 \\ x_2 = x_1, 0 \le x_2 \le 2 & \end{cases}$$

The binary tree algorithm uses the log of the variance matrix and sums the log values of x_2 moving from $x_1 = 0$ to $x_1 = 2$. When a significant change in the log sum occurs, a region boundary is determined. In this case, the log sum changes by a difference of 0.847 between $x_1 = 0.8$ and $x_1 = 1$ and by a difference of 1.386 from $x_1 = 1$ and $x_1 = 1.2$. A binary tree boundary can be established either at $x_1 = 1$ or $x_1 = 1.2$. Having divisions at both x_1 values is not desirable because there would be two boundaries as close as one grid distance. The binary tree algorithm determines where the boundary should be on the basis of an optimization function.

There are two considerations for an optimal binary tree: 1) variability of the tree regions; 2) the number and sizes of the tree regions. Of the many trees computed by the algorithm, the tree chosen is the one with the best balance between variability and complexity. The first step is to sum the mean variances of the two regions formed by making a tree division at $x_1 = 1$ as opposed to making a tree division at $x_1 = 1.2$. This approximation holds:

$$\sum \bar{s}_i^2 A_i \approx \bar{s}^2 A$$

where $\bar{s}_i^2$ is the mean variance in region i, A_i is the area of region i, $\bar{s}^2$ is the mean variance of the input space, and A is the area of the input space. The area is computed by multiplying the number of grid distances of length times the number of grid distances in width. Raising the $\bar{s}_i^2$ and $\bar{s}^2$ to the power 3/2, gives

$$\sum\left(\overline{s}_i^2\right)^{3/2} A_i \geq \left(\overline{s}^2\right)^{3/2} A$$

The ratio $\sum\left(\overline{s}_i^2\right)^{3/2} A_i \Big/ \left(\left(\overline{s}^2\right)^{3/2} A\right)$ increases as the differences in the $\overline{s}_i^2$ increase. Powers greater than 3/2 would further increase the differences in these two sums. The power of 3/2 was selected based on experimentation with several functions. In this example, the first division at $x_1 = 1$ with two areas of 50 grid distances squared has a sum $\sum\left(\overline{s}_i^2\right)^{3/2} A_i \Big/ \left(\overline{s}^2\right)^{3/2} = 116.359$. The second division at $x_1 = 1.2$ with two areas of 40 and 60 grid distances squared has a sum $\sum\left(\overline{s}_i^2\right)^{3/2} A_i \Big/ \left(\overline{s}^2\right)^{3/2} = 124.456$. Here, $1\Big/\left(\overline{s}^2\right)^{3/2}$ is used as a normalizing factor. This indicates, by this criterion, that the division at $x_1 = 1.2$ may be the better choice. However, we need to consider tree complexity which involves the number of regions and the sizes of regions. A smaller number of similarly sized regions may represent a function better than a large number of regions with different sizes. The trade off can be determined by computing a penalty for complexity. The penalty assessed for different size regions and the number of regions is given by

$$Penalty = \sum_{i,j} \left|A_i - A_j\right| \Big/ 2 + 2\left(n_{regions} - 1\right)$$

This penalty is subtracted from $\sum\left(\overline{s}_i^2\right)^{3/2} A_i \Big/ \left(\overline{s}^2\right)^{3/2}$ to get the optimal value. For the first tree boundary with the division $x_1 = 1$, the penalty becomes $Penalty = \sum_{i,j} \left|A_i - A_j\right| \Big/ 2 + 2(2-1) = 2$ since $\left|A_i - A_j\right| = 0$. For the second tree boundary with the division at $x_1 = 1.2$, the penalty becomes $Penalty = \sum_{i,j} \left|A_i - A_j\right| \Big/ 2 + 2(2-1) = 22$ since $\left|A_i - A_j\right| = 20$. With the penalty subtracted, the optimum values become 114.359 and 102.456, respectively. So, by including this penalty, the division at $x_1 = 1$ is optimal. Different weights could be applied to the values for $\sum\left(\overline{s}_i^2\right)^{3/2} A_i \Big/ \left(\overline{s}^2\right)^{3/2}$ and the *Penalty* to get different results. We found, based on several functions, that equal weights for both provided trees for several different functions that were reasonable.

The algorithm for dividing the input space into binary trees supplies many different trees. It does so by looping through a vector of lower cutoff values. Then, for each lower cutoff value it loops through a vector of step sizes. For each cutoff value and step size a binary tree is computed. Each binary tree created is assessed for its optimal value. The one with the highest optimal value is the one selected. The details of this are discussed in Subsection 2.3.

2.3. Binary Tree Algorithm in Detail

The binary tree algorithm is composed of five functions: **optimize.tree.regions, get.tree.regions, split, split.region,** and **get.optimum.value**.

The main function **optimize.tree.regions** is called with the input space dimension (1 or 2), the variance matrix, and input region size. It does the following steps:

- 1) Computes the logs of the variance matrix maximum and minimum. Sets the maximum lower cutoff to the maximum log of the variance matrix. Sets

the minimum lower cutoff to the maximum of the minimum log of the variance matrix or the maximum lower cutoff minus 10. Computes a vector of lower cutoff values by incrementing from the minimum lower cutoff to the maximum lower cutoff. Does the following for each lower cutoff:

- 2) Sets a minimum step size (δ) and a maximum step size. The maximum step size is half the maximum lower cutoff minus the minimum lower cutoff. Computes a vector of step sizes by incrementing from the minimum step size to the maximum step size. Loops through all step sizes of this vector for the current lower cutoff.
- 3) Calls **get.tree.regions** with input space dimension, variance matrix, lower cutoff, and step size, which returns the tree regions.
- 4) Calls **get.optimum.value** with the input space dimension, tree leaf regions, variance matrix. It returns the optimum value for the tree.
- 5) If the optimum value returned is larger than any previous value, stores this optimum value and the associated tree regions.
- 6) When processing is completed, returns the regions with the optimum value to the main program.

The function **get.tree.regions** is called with the input space dimension, variance matrix, lower cutoff, and step size. It does the following steps:

- 1) Initializes the tree regions matrix with the grid size of the variance matrix. Randomly selects a split direction if it is a two-dimensional space. Initializes the whole input space as a binary tree node with starting and ending indices for each dimension. Sets the region count to 1.
- 2) Calls **split** with the input space dimension, current region index, tree regions, variance matrix, cutoff, and step size.
- 3) If a split for the current region does not occur, changes split direction (if two dimensional) and calls **split** again.
- 4) Increments the region count. If region count in greater than the regions constructed (no new regions occurred with the last calls to **split**), ends split processing.
- 5) With the regions constructed, extracts leaf regions from the tree regions and returns the leaf regions to **optimize.tree.regions**.

The function **split** is called with input space dimension, current region index, tree regions, variance matrix, lower cutoff, step size. It does the following:

- 1) Sets the starting, ending and extents for the current region indices.
- 2) Calls **split.region** with input space dimension, current region index, tree regions, current split direction, starting, ending, and extents for the current region, lower cutoff, step size.
- 3) Returns the current tree regions to **get.tree.regions**.

The function **split.region** is called with the input space dimension, current region index, tree regions, current split direction, starting, ending and extents for the current region, variance matrix, lower cutoff, step size. It does the following:

- 1) Computes the sums for the trace of the variance matrix in the specified direction for the current region.
- 2) Computes the logs of these trace sums and gets the maximum log of these sums. Smooths the log sums of the trace using R's "lowess" function.
- 3) If the trace extents are ≥5 and the maximum log sum > lower cutoff, determines if there is a step size difference between adjacent log sums. If such a difference exists, it splits the region at that index where the difference occurs provided the difference occurs after (starting index +1) and before the (ending index −1).
- 4) Sets the two new regions to leafs and the split region to a node. Sets the two new leaf split directions opposite to the direction of the split region.
- 5) Returns the tree regions including any newly split regions to **split**.

After the tree regions are determined, the function **get.optimum.value** is called with input space dimension, regions (only leaf regions at this point) and variance matrix. It does the following:

- 1) Determines the means of the variance matrix for each leaf.
- 2) Sums the means raised to the 3/2 power times the tree region areas/lengths divided by the mean of the input space variance raised to the power 3/2. This result is component one of the optimal value.
- 3) Computes the sum of the differences in area/length of all region pairs and adds to this sum two times the number of regions minus one to get component two. (Note: This quantity measures the complexity of the tree.)
- 4) Returns the optimum value obtained by subtracting component two from component one to **optimize.tree.regions**.

2.4. Tree Region Processing

Each tree region is processed individually using the Metropolis-Hastings algorithm to get its individual process variance, σ^2, range parameter(s), θ_1 and θ_2 if two dimensional, and nugget, τ. The posterior distribution for each region is given as

$$f\left(\sigma^2, \theta_1, \theta_2, \tau \mid \boldsymbol{x}_1^t, \boldsymbol{x}_2^t, \boldsymbol{y}^t\right)$$
$$\propto \left(\sigma^2\right)^{-n/2} \left|\boldsymbol{R}\left(\theta_1, \theta_2\right)\right|^{-1/2} \exp\left(-\frac{1}{2\sigma^2}\left(\boldsymbol{y}^t\right)^{\mathrm{T}} \boldsymbol{R}\left(\theta_1, \theta_2\right)^{-1} \boldsymbol{y}^t\right) f\left(\theta_1\right) f\left(\theta_2\right)$$

where

$$\boldsymbol{R}\left(\theta_1, \theta_2\right)_{i,j} = \exp\left(-\left[\frac{\left(\boldsymbol{x}_{1,i}^t - \boldsymbol{x}_{1,j}^t\right)^2}{\theta_1} + \frac{\left(\boldsymbol{x}_{2,i}^t - \boldsymbol{x}_{2,j}^t\right)^2}{\theta_2}\right]\right)$$

and

$$\boldsymbol{R}\left(\theta_1, \theta_2\right)_{i,i} = 1 + \tau$$

where $\tau = nugget$.

The posterior distribution, except for the priors for θ_1 and θ_2, is a standard Gaussian process with a 0 mean. The priors are determined for four types of

functions: 1) irregular, 2) variable, 3) smooth, and 4) very smooth. (The user estimates the type of function at the beginning of processing when the variance grid and training point perspectives are displayed.) There are four considerations in the choice of the range priors:

- The grid distance is prominent in the prior selection. Grid points (also referred to as training points) near the point being predicted are the most influential in determining the predicted point. So the range parameters are often close to this distance squared except for regions where the functions are smooth or very smooth. When the function values are relatively small with little variation, the best predictions often use only the very nearest grid points.
- For cases where tree regions have variances much smaller than the region with the maximum process variance, it was discovered that an exponential prior with a mean of $1/\lambda$ dominates their range values.
- There are two classes of smooth functions: 1) smooth and 2) very smooth. These functions need larger range values and smaller nuggets to increase the accuracy of the emulator.
- Irregular functions require larger nuggets since actual points are scattered discontinuously through the function domain.

These considerations lead to the following priors for θ for each type of functions. The prior for θ for all but very smooth functions is:

$$f\left(\theta \mid \sigma_{\max}^2, \sigma^2, d_{grid}^2, K_{\min}, K_{\max}\right) = \lambda \exp\left(-\lambda\theta\right) \frac{\beta^{\alpha}}{\Gamma\left(\alpha\right)} \theta^{\alpha-1} \exp\left(-\beta\theta\right)$$

We will proceed to define the parameters and constants appearing in this prior.

- $\lambda = \left(\frac{\sigma_{\max}^2}{\sigma^2}\right)^{1/4}$ where $\sigma_{\max}^2$ is the maximum process variance of the tree regions and σ^2 estimates the process variance of the current tree region.
- $\beta = 1/\left(K_{prior}^2 d^2\right)$ where K_{prior} is either $K_{\min}$ or $K_{\max}$ or a value in between these constants. The computation of K_{prior} will be discussed last.
- The parameter d^2 is related to λ and the distance between grid points, d_{grid}, by the formula $d^2 = d_{grid}^2 / \lambda^{0.44}$.
- The parameter $\alpha = 1/K_{prior}^2$ implies that the Gamma prior has a mean of d^2.

This leaves K_{prior} to be defined. Two constants are chosen that bound K_{prior}. They are $K_{\min} = 0.075$ and $K_{\max} = 1$. When a function is estimated as smooth, K_{prior} is set to $K_{\max}$ for both θ_1 and θ_2. Also, for a correlated variable, K_{prior} is set to $K_{\max}$. When the function being estimated is variable or irregular, K_{prior} is given by the following formula which can only be justified empirically.

$$K_{prior} = K_{\min} + \frac{K_{\max} - K_{\min}}{7}\left(\lambda - 1\right)$$

If $K_{prior} \geq K_{\max}$, then $K_{prior} = K_{\max}$

If $K_{prior} \leq K_{\min}$, then $K_{prior} = K_{\min}$

What this says is when λ is close to 1, $K_{prior} = K_{\min}$ and the θ values are dominated by the Gamma prior. As the current process σ^2 gets smaller compared to $\sigma^2_{\max}$, λ gets larger and the exponential prior with λ dominates.

For functions estimated to be very smooth, the prior for θ_1 and θ_2 is:

$$f(\theta) = Gamma\left(\alpha = w^2 \times 1000, \beta = 1000\right)$$

where the mean of the *Gamma* function is w^2, the square of the size of the domain, and its standard variance is $w^2/1000$. This allows the ranges to include most of the whole domain.

After each tree region is processed to get their parameters σ^2, θ_1, θ_2, and τ, the parameters are smoothed (except for τ which is set to the average of τ's for the regions) at the region interfaces. Smoothing has been discussed previously in Subsection 2.1.

2.5. Prediction and Error Analysis

The smoothed parameters for the whole input space are now used to construct the covariance matrices for each predicted point. Standard Kriging formulas are then used to predict the point and its standard deviation. The formulas for prediction of the point $\left(\boldsymbol{x}^p_{1,i}, \boldsymbol{x}^p_{2,j}\right)$ are:

$$\boldsymbol{y}\left(\boldsymbol{x}^p_{1,i}, \boldsymbol{x}^p_{2,j}\right) = \boldsymbol{r}\left(\boldsymbol{x}^p_{1,i}, \boldsymbol{x}^p_{2,j}\right)^{\mathrm{T}} \boldsymbol{R}^{-1}\left(\boldsymbol{x}^p_{1,i}, \boldsymbol{x}^p_{2,j}\right) \boldsymbol{y}^t$$

where

$$\boldsymbol{R}\left(\boldsymbol{x}^p_{1,i}, \boldsymbol{x}^p_{2,j}\right)_{k,l} = \exp\left(-\left[\left(\boldsymbol{x}^t_{1,k} - \boldsymbol{x}^t_{1,l}\right)^2 \Big/ \theta_1\left(\boldsymbol{x}^p_{1,i}, \boldsymbol{x}^p_{2,j}\right) + \left(\boldsymbol{x}^t_{2,k} - \boldsymbol{x}^t_{2,l}\right)^2 \Big/ \theta_2\left(\boldsymbol{x}^p_{1,i}, \boldsymbol{x}^p_{2,j}\right)\right]\right)$$

$$\boldsymbol{R}\left(\boldsymbol{x}^p_{1,i}, \boldsymbol{x}^p_{2,j}\right)_{k,k} = 1 + \tau$$

and

$$\boldsymbol{r}\left(\boldsymbol{x}^p_{1,i}, \boldsymbol{x}^p_{2,j}\right)_{k} = \exp\left(-\left[\left(\boldsymbol{x}^p_{1,i} - \boldsymbol{x}^t_{1,k}\right)^2 \Big/ \theta_1\left(\boldsymbol{x}^p_{1,i}, \boldsymbol{x}^p_{2,j}\right) + \left(\boldsymbol{x}^p_{2,j} - \boldsymbol{x}^t_{2,k}\right)^2 \Big/ \theta_2\left(\boldsymbol{x}^p_{1,i}, \boldsymbol{x}^p_{2,j}\right)\right]\right)$$

In the formulas above, the subscripts i, j are for the predicted point and the subscripts k, l are for the training point data. For the error prediction (σ) we have:

$$\boldsymbol{\sigma}^2\left(\boldsymbol{x}^p_{1,i}, \boldsymbol{x}^p_{2,j}\right)_{point}$$

$$= \boldsymbol{\sigma}^2\left(\boldsymbol{x}^p_{1,i}, \boldsymbol{x}^p_{2,j}\right)_{process}\left(1 + \tau - \boldsymbol{r}\left(\boldsymbol{x}^p_{1,i}, \boldsymbol{x}^p_{2,j}\right)^{\mathrm{T}} \boldsymbol{R}\left(\boldsymbol{x}^p_{1,i}, \boldsymbol{x}^p_{2,j}\right)^{-1} \boldsymbol{r}\left(\boldsymbol{x}^p_{1,i}, \boldsymbol{x}^p_{2,j}\right)\right)$$

$$\boldsymbol{\sigma}\left(\boldsymbol{x}^p_{1,i}, \boldsymbol{x}^p_{2,j}\right)_{point} = \sqrt{\boldsymbol{\sigma}^2\left(\boldsymbol{x}^p_{1,i}, \boldsymbol{x}^p_{2,j}\right)_{point}}$$

For a one dimensional input space, the formulas are simplified since there is only one vector, $\boldsymbol{x}_1$, one range vector, $\boldsymbol{\theta}_1$, and one standard deviation vector $\boldsymbol{\sigma}$.

3. Illustrations of the Improved Treed Gaussian Process

Two illustrations of the Improved Treed Gaussian process are discussed in this

section and their results compared to the Treed Gaussian process.

3.1. First Illustration of the Improved Treed Gaussian Process

Here we demonstrate how the Improved Treed Gaussian process (abbreviated Improved TGP) produces better predictions and error estimates than the Treed Gaussian process (abbreviated TGP) for the variable function shown in Figure 5.

The equation of the function is

$$f(x_1, x_2) = (4 * x_1 - 2) * \exp\left(-(4 * x_1 - 2)^2 - (4 * x_2 - 2)^2\right)$$

where $0 \le x_1 \le 2$ and $0 \le x_2 \le 2$.

For this function, a 10 by 10 grid of training points is computed. (This grid size is chosen by the user.) Next the variance matrix is computed. Figure 6 shows perspectives of the variance matrix and the training points for the function. Then the binary tree algorithm is run to get the binary tree shown in Figure 7.

Each region is now processed with the Metropolis-Hastings algorithm to get its process variance (σ^2), ranges (θ_1, θ_2), and nugget τ. The smoothing is done on the prediction grid. In this case, the prediction grid is a 41 by 41 grid of points. (This is chosen by the user based on what size he/she thinks represents the function best.)

The last step is the prediction of these gridded data points and their standard deviations. A covariance matrix is computed for each predicted point and the standard kriging formulas discussed in Section 2.5 are used to compute the point estimates and their error estimates, σ. Figure 8 illustrates the perspective of the predicted points, the perspective of the actual point values, the perspective of the standard deviations (error estimates), and the perspective of the actual errors.

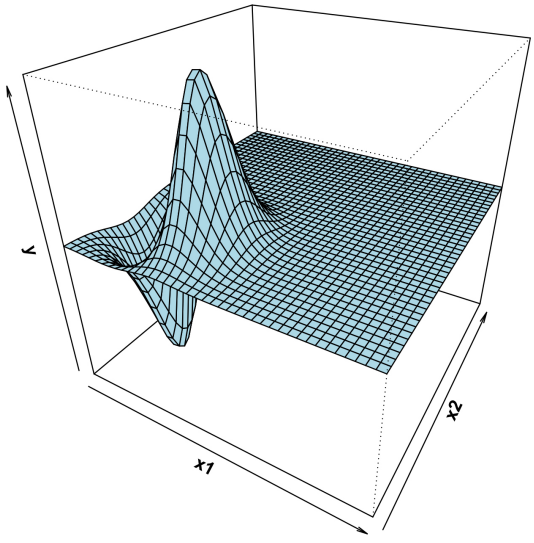

Figure 5. Perspective of first illustrated function: this function has one quadrant with variation while the other three quadrants have very little variation.

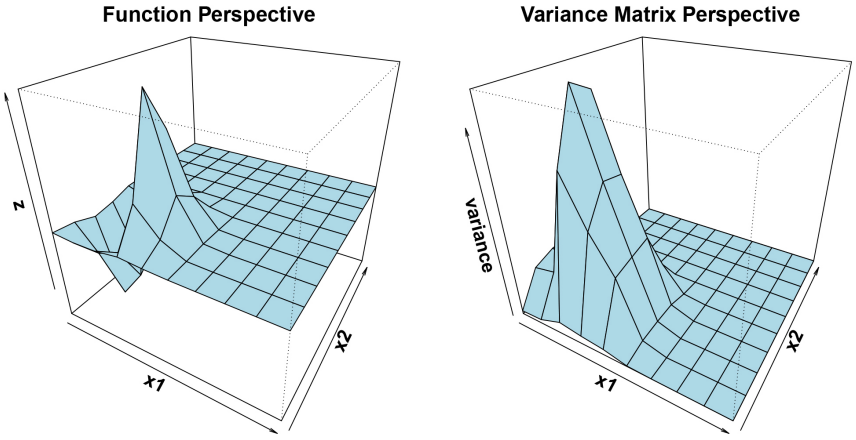

Figure 6. Perspectives for first illustration of the variance matrix and training points: the perspectives of both the variance matrix and the training points, which give a rough view of the function, indicate one region of variance and an L-shaped region of very little variance. A binary tree of size three is computed.

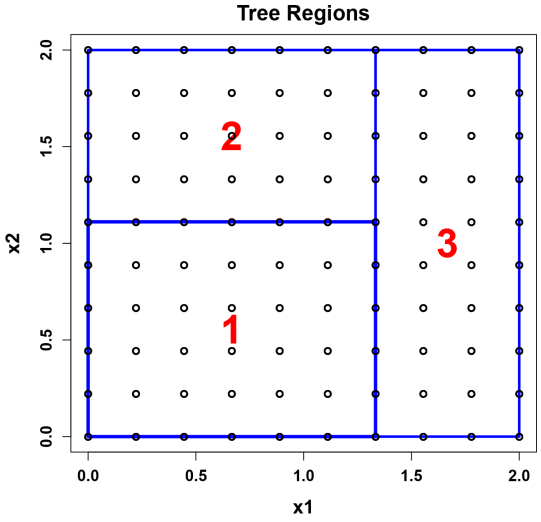

Figure 7. Binary tree for the first illustration: the binary tree has three leafs numbered in the order of their approximate process variance. The black circles are the training points.

Comparing the results for our Improved TGP with those for TGP, Figure 9 shows TGP results for the perspective of the predicted points, the perspective of the actual point values, the perspective of the standard deviations, and the perspective of the actual errors. Notice that on the side of the region with variation, the predicted surface varies from the actual surface. There is a little wave which is sort of like a speed bump. Also, it can be seen that there is a large difference in TGP estimated errors (σ) and its actual errors. The Improved TGP estimated errors are more closely related to its actual errors.

The main difference between our emulator results and TGP lies in the errors and error estimates. A simulation study of the errors and error estimates was done using 100 runs of both the Improved TGP and TGP. Table 1 shows the Improved TGP models the surface better than TGP. The Improved TGP mean

absolute error is much smaller than that of TGP. The maximum absolute error for Improved TGP is smaller by nearly a factor of 4 and the estimated errors are smaller for the Improved TGP than for TGP. Also of interest, the Improved TGP had 0 points with absolute error to σ ratios > 2.5 while TGP had an average of 6.07 points with ratios > 2.5. We note that the standard deviations for all these

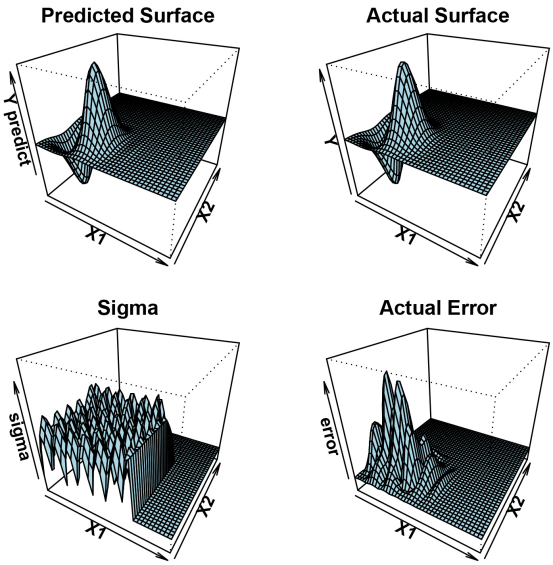

Figure 8. Perspectives of the predicted points and estimated errors compared to the actual function and actual errors: the perspective on the upper left has the predicted point values, the perspective on the upper right has the actual point values, the perspective on the lower left is the estimated errors (σ) and the perspective on the lower right is the actual errors.

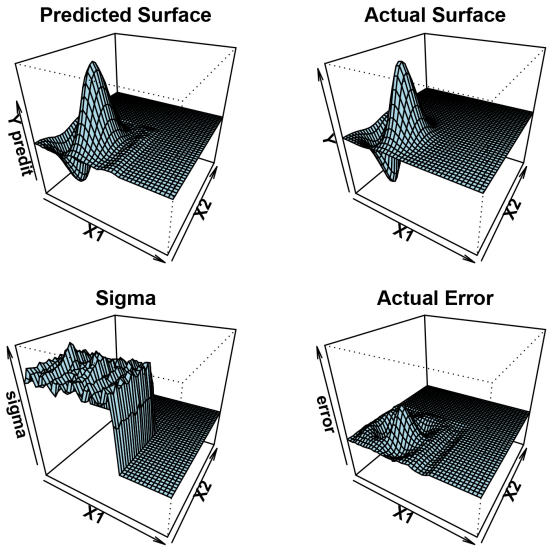

Figure 9. Perspectives of the predicted points and estimated errors compared to the actual function and actual errors: the perspective on the upper left has the predicted point values, the perspective on the upper right has the actual point values, the perspective on the lower left is the estimated error (σ) and the perspective on the lower right is the actual error. Notice that the estimated errors are much larger than the actual errors.

Table 1. Table comparing the improved TGP and TGP results: the first row has our Improved TGP data and the second row has the TGP data. Each statistic is the mean for the 100 runs. The last column is the mean number of predicted points with $|error|/\sigma > 2.5$.

Process	Mean Absolute Error	Mean Max Absolute Error	Mean σ	Mean Max σ	Mean Points
Improved TGP	0.00308	0.06018	0.01302	0.04693	0
TGP	0.01223	0.23351	0.10559	0.42050	6.07

statistics are considerably smaller for the Improved TGP compared to TGP conveying the superior stability of the Improved TGP.

3.2. Second Illustration of the Improved Gaussian Process

Here we demonstrate how the Improved TGP produces better predictions and error estimates than TGP for the variable function shown in the perspective view (Figure 10).

For this function, x_1 and x_2 are translated and rotated by 45˚ to get the variables u and v. The function is then expressed in terms of u and v. In these variables, it is similar to the first function illustrated. The equations for the function are:

$$u = x_1 \cos(\pi/4) - x_2 \sin(\pi/4) + 1/2$$

$$v = x_1 \sin(\pi/4) + x_2 \cos(\pi/4) + 1/2$$

$$f(x_1, x_2) = (4*u-2)*\exp\left(-(4*u-2)^2 - (4*v-2)^2\right)$$

where $0 \le x_1 \le 2$ and $0 \le x_2 \le 2$.

For this function a 10 by 10 grid of training points is computed. Next the variance matrix is computed. Figure 11 shows the perspectives of the variance matrix and the training points for the function. Then a binary tree algorithm is run to get the binary tree in Figure 12.

Each region is now processed with the Metropolis-Hastings algorithm to get its process variance (σ^2), ranges (θ_1, θ_2), and nugget τ. The smoothing is done on a prediction grid of 41 by 41 points. The last step is the prediction of these gridded data points and their standard deviations. A covariance matrix is computed for each predicted point and the standard Kriging formulas discussed in Section 2 are used to compute the point estimates and their error estimates, σ. Figure 13 shows the perspective of the predicted points, the perspective of the actual point values, the perspective of the standard deviations (estimated errors), and the perspective of the actual errors.

Comparing the results with what we get from TGP, Figure 14 shows the TGP results for the perspective of the predicted points, the perspective of the actual point values, the perspective of the standard deviations, and the perspective of the actual errors.

There are some differences in the TGP surface and the actual surface around the edges of the variable region. However, the main difference compared to the

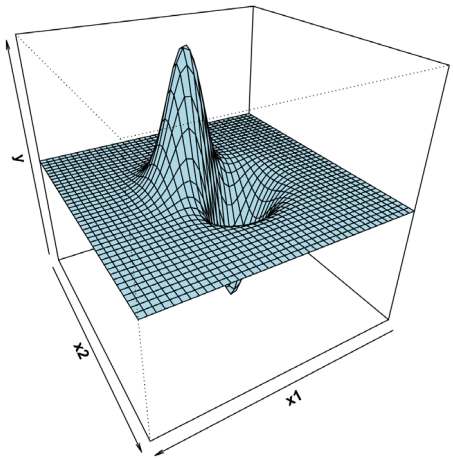

Figure 10. Perspective of second illustrated function: this function has variation in the center region of the input space while around the edges of the input space it has little variation.

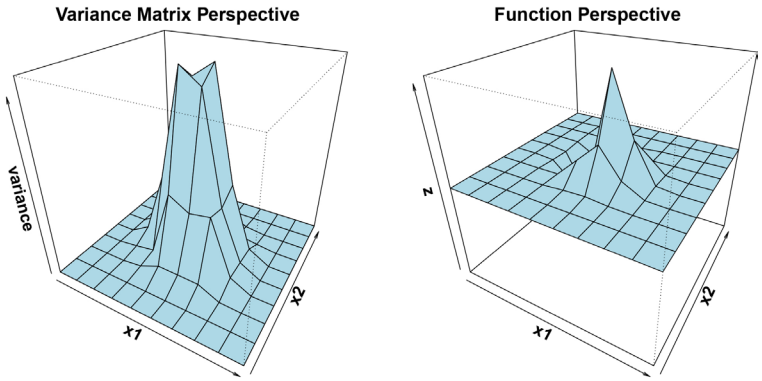

Figure 11. Perspectives for second illustration of the variance matrix and training points: the perspectives of both the variance matrix and the training points, which give a rough view of the function, has a central region of variance surrounding by a region of very little variance.

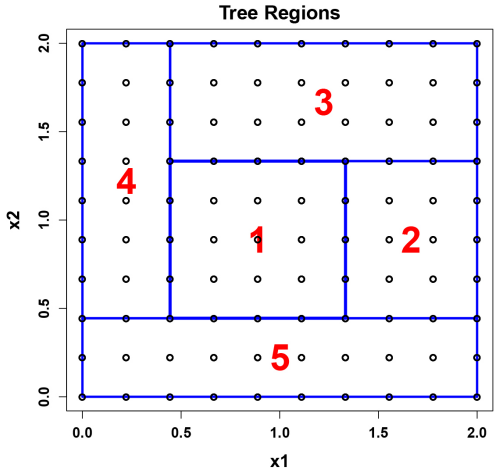

Figure 12. Binary tree for the second illustration: the binary tree has five leafs numbered in the descending order of their approximate process variance. The black circles are the training points.

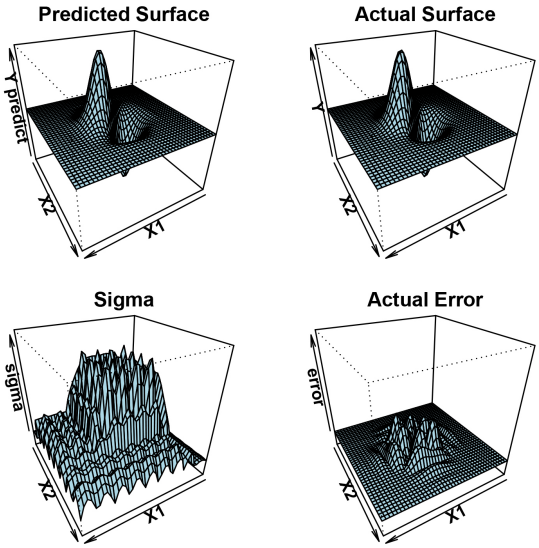

Figure 13. Perspectives of the predicted points and estimated errors compared to the actual function and actual errors: the perspective on the upper left has the predicted point values, the perspective on the upper right has the actual point values, the perspective on the lower left is the estimated error (σ) and the perspective on the lower right is the actual error.

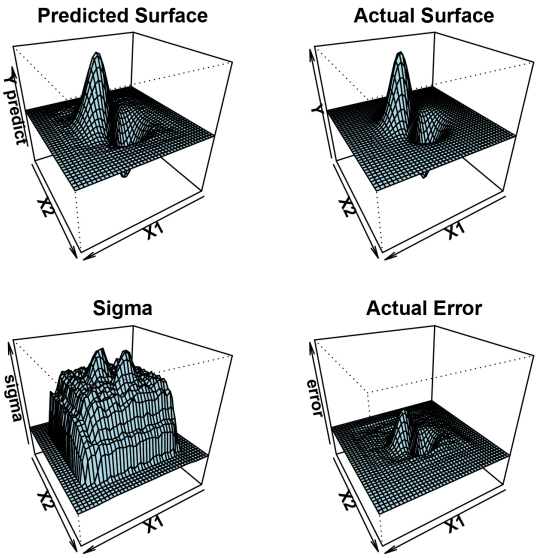

Figure 14. Perspectives of the predicted points and estimated errors compared to the actual function and actual errors for TGP: the perspective on the upper left has the predicted point values, the perspective on the upper right has the actual point values, the perspective on the lower left is the estimated actual error (σ) and the perspective on the lower right is the actual error.

Improved TGP lies in the errors and error estimates, σ. A comparison of the errors and error estimates was done for 100 runs of both the Improved TGP and TGP. Table 2 shows the Improved TGP models the surface better than TGP. The Improved TGP mean absolute error is much smaller than that of TGP. The maximum absolute error for the Improved TGP is smaller by about a factor of 10.

Table 2. Table comparing the improved TGP and TGP results: The first row has the improved TGP data and the second row has the TGP data. Each statistic is the mean for 100 runs. The last column is the mean number of predicted points with $|error|/\sigma > 2.5$.

Process	Mean Absolute Error	Mean Max Absolute Error	Mean σ	Mean Max σ	Mean Points
Improved TGP	0.00162	0.01994	0.01598	0.05310	0
TGP	0.01675	0.26825	0.15457	0.50168	7.87

Also, the estimated errors are smaller for the Improved TGP than for TGP, and, the Improved TGP had 0 points with absolute error to σ ratios > 2.5 while TGP had an average of 7.87 points with ratios > 2.5. The standard deviations for all these statistics are considerably smaller for the Improved TGP over the TGP conveying the superior stability of the Improved TGP.

4. Analysis of Motorcycle Stopping Data with Improved Treed Gaussian Process

Here we compare the Improved TGP predictions and error estimates to those of the TGP for the function represented by the data of time versus acceleration for the stopping of a motorcycle. The data from [12] has been modified to have increments of 0.2 seconds. The modified data is shown in **Figure 15**. Where more than one data point existed in a time increment, the average of these points is used. For increments where no data existed, the acceleration has been determined as a weighted average of the closest time accelerations.

This irregular function has no associated analytic function. For this irregular function a grid of 19 data points is used. Next the variance matrix is computed. Below, plots are shown in **Figure 16** of the variance matrix and the training points for a 19 point grid of the irregular function's data.

The binary tree algorithm returned a binary tree with one leaf. The input space is now processed with the Metropolis-Hastings algorithm to get its process variance σ^2, range θ_1, and nugget, τ. No smoothing is required for one leaf. Next the 265 data points are predicted along with their standard deviations. A single covariance matrix is computed for all data points predicted. **Figure 17** shows the plot of the predicted points, the plot of the actual point values, the plot of the standard deviations, and the plot of the actual errors.

For comparison to the Improved TGP, **Figure 18** shows the plots of the predicted points, the actual point values, the standard deviations, and the actual errors for TGP.

The predicted surfaces of the Improved TGP and TGP are very similar. Our Improved TGP and TGP were run 100 times to get the data in **Table 3**. On average, the Improved TGP has a lower value for the absolute mean error and absolute maximum error. Furthermore, the mean for σ is a closer estimate of the mean absolute error. The means for the number of points with ratios of absolute error to σ greater than 2.5 are both 0.

Table 3. Table comparing improved treed gaussian process and TGP results: the first row has the improved TGP data and the second row has the TGP data. Each statistic is the mean for 100 runs. The last column is the mean number of predicted points with $|error|/\sigma > 2.5$.

Process	Mean Absolute Error	Mean Max Absolute Error	Mean σ	Mean Max σ	Mean Points
Improved TGP	14.23	80.98	40.79	41.36	0
TGP	15.78	89.29	71.48	83.55	0

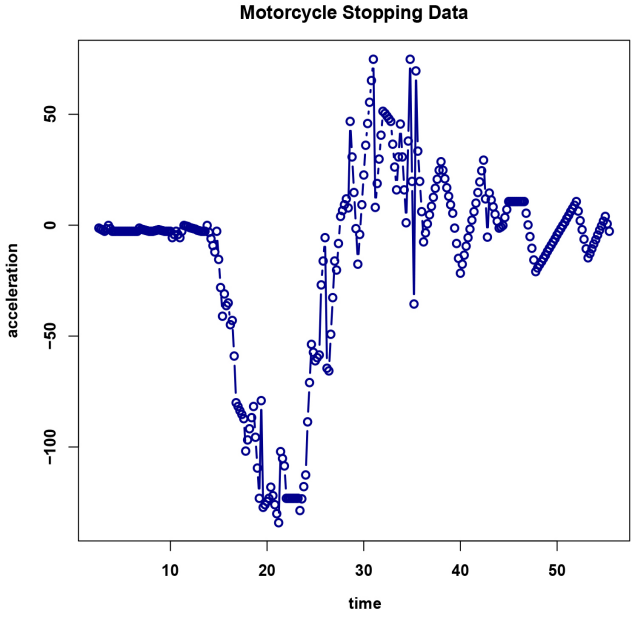

Figure 15. Plot of one dimensional motorcycle stopping data: this represents an irregular function with the first portion being uniform speed followed by abrupt deceleration. The last portion slowly approaches zero acceleration.

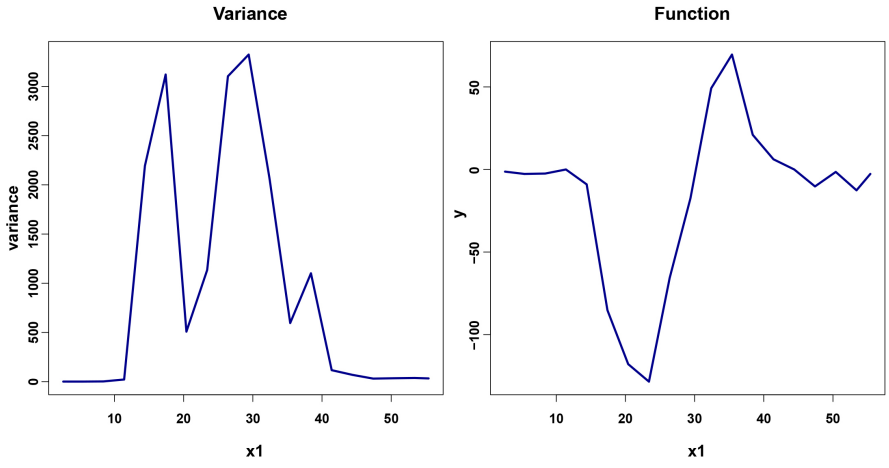

Figure 16. Plots for motorcycle stopping variance matrix and data points: the plots of both the variance matrix and the training points, which give a rough view of the function, reveal an irregular variable function.

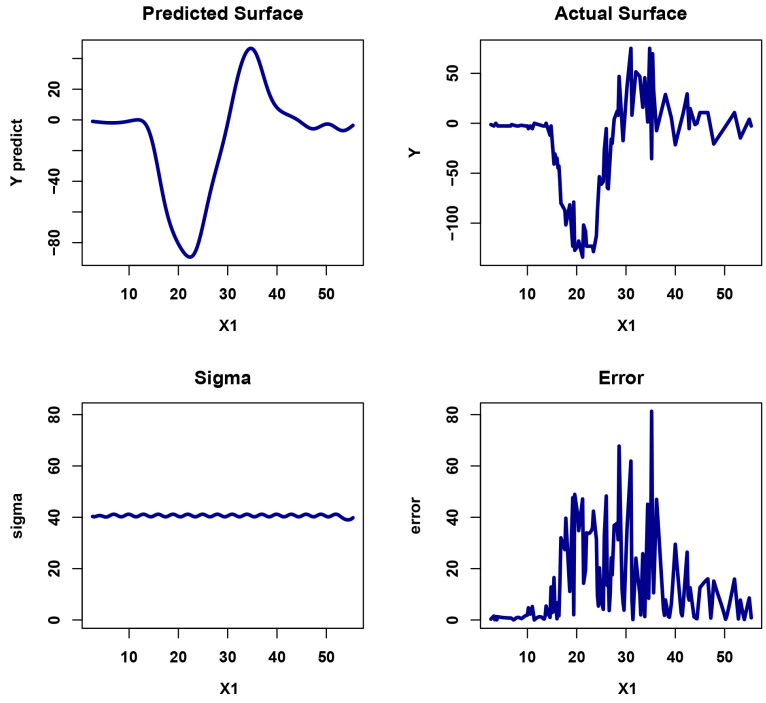

Figure 17. Plots of the predicted points and estimated errors compared to the actual function and actual errors: the plot on the upper left has the predicted point values, the plot on the upper right has the actual point values, the plot on the lower left is the estimated error (σ) and the plot on the lower right is the actual error.

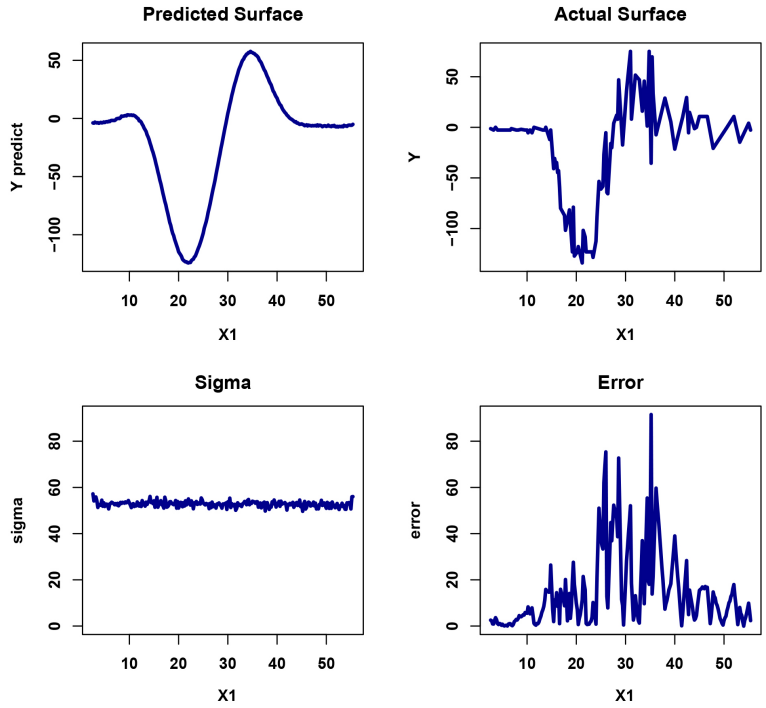

Figure 18. Plots of the predicted points and estimated errors compared to the actual function and actual errors for TGP: the plot on the upper left has the predicted point values, the plot on the upper right has the actual point values, the plot on the lower left is the estimated error (σ) and the plot on the lower right is the actual error.

5. Conclusions

We propose an improved TGP for efficiently modeling functions with regions of different variability.

- It has been shown that the improved TGP gives better results than TGP for low-dimensional functions with regions of different variability.
- The predictions of points and their associated standard deviations are stable. TGP can vary considerably from one call to another for the same target function and training points.
- The improved TGP has better smoothing between the predicted points in different tree regions than TGP.
- For functions with multiple regions, there is a different covariance matrix for each predicted point providing predictions with smaller actual errors and error estimates more compatible with those actual errors than TGP.
- Although processing time is greater, processing time can be benefited by making use of a more accurate and efficient matrix inverter.

Future research needs to be done to extend the binary tree algorithm for higher dimensional applications.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Gramacy, R. and Lee, H.K. (2008) Bayesian Treed Gaussian Process Models with Application to Computer Modeling. Journal of the American Statistical Association, **103**, 1119-1130. https://doi.org/10.1198/016214508000000689

[2] Santner, T., Williams, B. and Notz, W. (2003) The Design and Analysis of Computer Experiments. Springer, New York. https://doi.org/10.1007/978-1-4757-3799-8

[3] Kleijnen, J.P.C. (2015) Design and Analysis of Simulation Experiments. 2nd Edition, Springer, New York.

[4] Higdon, D., Swall, J. and Kern, J. (1999) Non-Stationary Spatial Modeling. *Bayesian Statistics*, **16**, 761-768.

[5] Paciorek, C.J. and Schervish, M. (2006) Spatial Modelling Using a New Class of Nonstationary Covariance Functions. *Environmentrics*, **17**, 483-506. https://doi.org/10.1002/env.785

[6] Liang, W.W.J. and Lee, H.K.H. (2019) Bayesian Nonstationary Gaussian Process Models for Large Datasets via Treed Process Convolutions. *Advances in Data Analysis and Classification*, 1-22.

[7] Kim, H., Mallick, B. and Holmes, C. (2005) Analyzing Nonstationary Spatial Data Using Piecewise Gaussian Processes. *American Statistical Association*, **100**, 653-668. https://doi.org/10.1198/016214504000002014

[8] Heaton, M., Christensen, W. and Terres, M. (2014) Nonstationary Gaussian Process Models Using Spatial Hierarchical Clustering from Finite Differences. *Technometrics*, **59**, 93-101. https://doi.org/10.1080/00401706.2015.1102763

[9] Li, F. and Sang, H. (2019) Spatial Homogeneity of Regression Coefficients for Large

Datasets. *American Statistical Association*, **114**, 1050-1062. https://doi.org/10.1080/01621459.2018.1529595

[10] Konomi, B., Sang, H. and Mallick, B. (2014) Adaptive Bayesian Nonstationary Modeling for Large Datasets Using Covariance Approximations. *Computational and Graphical Statistics*, **23**, 802-829. https://doi.org/10.1080/10618600.2013.812872

[11] Kim, H., Mallick, B. and Holmes, C. (2005) Analyzing Nonstationary Spatial Data Using Piecewise Gaussian Processes. *Journal of the American Statistical Association*, **100**, 653-668. https://doi.org/10.1198/016214504000002014

[12] Gramacy, R. (2005) Bayesian Treed Gaussian Process Models. PhD Thesis, UC Santa Cruz.

Appendix A—Pseudo Code for Improved Treed Gaussian Process and Binary Tree Functions

Pseudo Code for the Improved Treed Gaussian Process:

Compute grid of training points and the associated variance matrix.

Set flag to process the function as irregular, variable, smooth, or very smooth.

Call optimize.tree.regions to compute binary tree.

For each tree region run Metropolis-Hastings to compute ranges, process variance and nugget.

Smooth range parameters and process variance between regions.

Predict points and standard deviations for each point.

Pseudo Code for Binary Tree Algorithm Functions:

Call optimize.tree.regions with input space dimension (1 or 2), variance matrix, and input space size.

Compute the logs of the variance matrix maximum and minimum
and set an increment (δ) for the lower cutoffs.

Set the maximum lower cutoff to the log of the variance matrix maximum.

Set the minimum lower cutoff to the maximum of the log of the
variance matrix minimum or the maximum lower cutoff minus 10.

Make a lower cutoff vector by incrementing from the minimum lower
cutoff to the maximum lower cutoff.

For each lower cutoff value from the vector of lower cutoff values do the following:

Set a minimum step size (δ) and maximum step size and a step size increment (δ).

Note: Step size maximum is 1/2 times the maximum lower cutoff minus the minimum lower cutoff.

Make a step size vector by incrementing from the minimum step size to the maximum step size.

For each step size and current cutoff value do the following:

Call **get.tree.regions** with input space dimension, variance matrix, lower cutoff,
and step size, returning tree regions.

Call **get.optimum.value** with the input space dimension, tree regions, variance matrix,
returning optimal value.

If optimal value returned is larger than any previous value, store optimum value and tree regions.

End step size loop.

End lower cutoff loop and return tree regions with optimal value to the main function.

Call get.tree.regions with input space dimension, variance matrix, lower cutoff, and step size.

Initialize the tree regions matrix with the grid size of the variance matrix.

Randomly select a split direction for a two dimensional input space.
Initialize the whole input space as a leaf.
Set the region count to 1.
While the tree regions are not complete:
Call **split** with input space dimension, region count, tree regions, variance matrix,
lower cutoff, and step size.
If a split does not occur, change split direction (if two dimensional) and call split again.
Increment region count.
If region count in greater than regions constructed, exit while loop.
With regions constructed, extract leaf regions from region matrix.
Return leaf regions as index limits to **optimize.tree.regions**.

Call split with input space dimension, region count, tree regions, variance matrix, lower cutoff, and step size.

Set the starting, ending and extents for the current region indices.
Call **split.region** with input space dimension, region count, tree regions, current split direction,
starting, ending, and extents for the current region, lower cutoff, and step size.
Return tree regions to **get.tree.regions**.

Call split.region with the input space dimension, region count, tree regions, current split direction, starting, ending and extents for the current region, variance matrix, lower cutoff, and step size.

Compute the sums for the trace of the variance matrix in the specified direction.
Compute the logs of these trace sums and get the maximum log of these sums.
Smooth the log sums of the trace using R's "lowess" function.
If the trace extents are ≥ 5 and the maximum log > lower cutoff:
Determine if there is a step size difference between region's adjacent log sums
occurring after (starting index +1) and before the (ending index −1).
If such a difference exists:
Split the region at that index where it occurs.
Set the two new regions to leafs and the split region to a node.
Set the two new regions split directions different from the split region,
if the input space is two dimensional.
Return tree regions to **split**.

Call get.optimum.value with input space dimension, regions(only leaf regions at this point) and variance matrix.

Determine the means of the variance matrix for each leaf.
Sum the means raised to the power 3/2 times the tree region areas/lengths di-

vided by the mean of

the input space raised to the power 3/2 to get component one of the optimal value.

Note: The mean of the input space raised to the power 3/2 is a normalizing factor.

Compute the sum of the differences in area/length of all region pairs

adding two times the number of regions minus one to get component two.

Note: This quantity measures the complexity of the tree.

Return the optimum value, component one minus component two, to **optimize.tree.regions**.

Notations

A—area of input space in grid distance squared units
A_i —area of input region i in grid distance squared units
$\overline{s}^2$ —average variance of input space
$\overline{s}_i^2$ —average variance of region i
Penalty—variable quantifying the complexity of a binary tree
x_1 —"x" coordinate for a one or two dimensional function being emulated
x_2 —"y"coordinate for a two dimensional function being emulated
$\boldsymbol{x}_1^t$ —vector of grid training points on "x" axis
$\boldsymbol{x}_2^t$ —vector of grid training points on "y" axis
$\boldsymbol{y}^t$ —function training point values
σ^2 —variance of a region's Gaussian process
$\sigma_{\max}^2$ —maximum variance for the regions
$\boldsymbol{\sigma}$ —matrix of standard deviations for input space point predictions
$\boldsymbol{\sigma}^2$ —matrix of variances for input space point predictions
λ —ratio of $\sigma_{\max}^2$ to region with σ^2 to 1/4 power
d_{grid} —grid distance: input space length divided by number of divisions
$\boldsymbol{R}(\theta_1,\theta_2)$ —covariance matrix
$\boldsymbol{r}(\theta_1,\theta_2)$ —vector of covariances used for kriging
θ_1 —range of variable x_1 for region's Gaussian process
θ_2 —range of variable x_2 for region's Gaussian process
$\boldsymbol{\theta}_1$ —matrix of x_1 range variables for input space point predictions
$\boldsymbol{\theta}_2$ —matrix of x_2 range variables for input space point predictions
τ —nugget for region's Gaussian process
$K_{\min}$ —constant lower limit for θ priors
$K_{\max}$ —constant upper limit for prior for θ priors
K_{prior} —constant for θ prior computed from $K_{\min}, K_{\max}, \lambda$
d^2 — $d_{grid}^2 / \lambda^{0.44}$ -used in prior function for θ's
β — $1/\left(K_{prior}^2 d^2\right)$ -used in prior function for θ's
α — $1/K_{prior}^2$ -used in prior function for θ's
w—length and width of input space
$\boldsymbol{x}_1^p$ —vector of predicted points on the "x" axis
$\boldsymbol{x}_2^p$ —vector of predicted points on the "y" axis
$\boldsymbol{y}^p$ —matrix of predicted point values