Latent map Gaussian processes for mixed variable metamodeling

Students :
Julien Crimotel
Lucas Laloue
Gaulthier Voland

Tutor :
Julien Pelamatti

Projet méthodologique
2023-2024

Contents

List of Figures

Lexicon

For a better comprehension of this work, some of the most useful terms are explained below.

- **Gaussian Processes:** A family of random variables which are Gaussian vectors if we take a finite number of them. Those Gaussian vectors have good mathematical properties, helping the computing of the model. As Gaussian vectors can be defined with its parameters: $\mu \in \mathbb{R}^d$ for its mean and $\Sigma \in \mathbb{R}^{d \times d}$ for its covariance (both in finite dimension), a GP can also be defined with its mean (m) and covariance (cov) functions.

- **Latent space:** It is a conceptual space where hidden or non observable factors are represented. For example, in Natural Language Processing, the models are only given characters and could interpret in the latent space interactions between the letters (to create words, phrases, etc.).

- **Regression:** It is a line drawn to best fit a cloud of points on a graph. It helps to summarize the overall trends over these points.

- **Map**: A function which transforms the type of the variables. In our specific case, the map functions are given qualitative inputs for quantitative outputs.

- **Kernel:** A measuring tool in machine learning. It helps quantify the similarity or dissimilarity between data points. It compares pairs of points and assigns a score based on their resemblance. This score influences how much these points should influence each other in a machine learning model.

- **Cross validation:** To better fit a model, different values of parameters should be tested. By cutting the dataset into "train" and "test" datasets a few times (often 5 times), we ensure the parameters are well-adjusted.

- **Mixed variable:** features that could be both qualitative (colors) and quantitative (temperatures, height, etc.).

- **Bayesian method – Prior – Posterior:** Unlike the frequentist method, the Bayesian one uses *prior* information. We use expert analysis to give the model more information, and the data is then used to adjust the experts' view, and finally it gives the *posterior* information.

- **Benchmark:** In the context of evaluating statistical models, a benchmark refers to a standard or reference point against which the performance of a given model can be measured. It serves as a baseline or a point of comparison to assess how well a particular statistical model performs in relation to established standards or other models. It involves a measure of computation time or accuracy for example.

- **Gradient descent:** Gradient descent is an optimization algorithm commonly used to minimize the cost or loss function in the context of machine learning and mathematical optimization problems. The goal is to find the minimum of a function iteratively by adjusting the parameters or weights of a model.

Notations

Furthermore, for all the paper, we decided, for a better comprehension of the paper, to write differently the numbers, vectors and matrices. We thus consider :

- $\mathbf{X}$ (bold and capital letter) is a matrix,
- $\mathbf{x}$ (bold and lowercase letter) is a vector,
- x is a number

Introduction

In certain supervised learning scenarios involving both classification and regression, Gaussian processes (GPs) emerge as a valuable tool due to their inherent properties derived from the normal distribution and their adeptness in modeling intricate relationships. However, GPs are inherently designed for quantitative data, posing a limitation when confronted with mixed variables. To address this limitation, Nicholas Oune and Ramin Bostanabad introduced latent map Gaussian processes (LMGPs) [4], specifically tailored for handling mixed variables.

A straightforward approach would be to encode each qualitative variable with the one-hot encoding method, but it becomes inefficient as the number of variables increases, and it overlooks potential correlations between features. An alternative strategy involves mapping each qualitative input to a latent space where distances can be calculated. This concept was developed by Zhang et al., leading to the creation of latent variable Gaussian processes (LVGPs) [4]. While LVGPs demonstrated superior performance compared to conventional methods, N. Oune and R. Bostanabad enhanced LVGPs with two key distinctions: a linear prior representation and the utilization of a unified latent space for all variables, as opposed to LVGPs that map each qualitative feature to a distinct latent space.

This paper elucidates the construction and implementation of LMGPs to effectively learn from mixed datasets. A comparative analysis against other state-of-the-art methods will ultimately showcase the efficacy of LMGPs.

1 Gaussian process modeling to LMGP

1.1 Gaussian processes

Gaussian Process (GP) modeling is a powerful technique for building meta-models, capturing complex relationships between inputs and outputs in a probabilistic framework. GPs are non-parametric and belong to the family of kernel methods. In GP modeling, we assume that the function we are trying to model is a draw from a multivariate Gaussian distribution [1][3].

It means that, given a set of input-output pairs $(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), ..., (\mathbf{x_n}, y_n)$, where x_i represents the input and y_i the corresponding output, a Gaussian Process (GP) defines a distribution over functions. Each function f is considered a draw from a multivariate Gaussian distribution. The GP is fully characterized by a mean function $m(\mathbf{x})$ and a covariance function (kernel) $k(\mathbf{x}, \mathbf{x}')$.

The GP is specified as follows:

$$f(x) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$$

This implies that for any finite set of inputs $\mathbf{X} = [\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_n}]$, the corresponding outputs $\mathbf{y} = [y_1, y_2, ..., y_n]^\intercal$ follow a joint multivariate Gaussian distribution:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(\mathbf{x_1}) \\ m(\mathbf{x_2}) \\ \vdots \\ m(\mathbf{x_n}) \end{bmatrix}, \begin{bmatrix} k(\mathbf{x_1}, \mathbf{x_1}) & k(\mathbf{x_1}, \mathbf{x_2}) & \cdots & k(\mathbf{x_1}, \mathbf{x_n}) \\ k(\mathbf{x_2}, \mathbf{x_1}) & k(\mathbf{x_2}, \mathbf{x_2}) & \cdots & k(\mathbf{x_2}, \mathbf{x_n}) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x_n}, \mathbf{x_1}) & k(\mathbf{x_n}, \mathbf{x_2}) & \cdots & k(\mathbf{x_n}, \mathbf{x_n}) \end{bmatrix} \right)$$

A GP can be seen as a function where each output $\mathbf{y}$ is a Gaussian (*cf.* "cake representation" figure 1.1.1). Here, a zero-mean GP is represented, meaning that the highest probability to observe a new point would be close to $y = 0$, for all input $\mathbf{x}$.

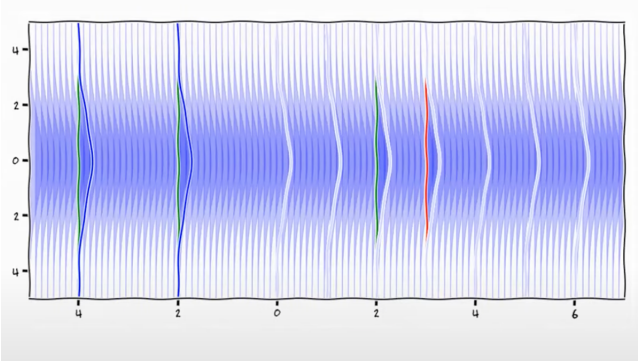

Figure 1.1.1: "Cake representation" of a GP
Credits: Carl Henrik Ek

1.2 The link with modeling problems

Now that we know what Gaussian processes are, one may ask what are they used for? Thanks to the natural properties of Gaussian densities, we can express conditional distribution of $\mathbf{y}_{test}$ given new observations $\mathbf{X}_{test}$. According to the model assumptions, the joint distribution of training and test points is given by:

$$\begin{bmatrix} \mathbf{y}_{train} \\ \mathbf{y}_{test} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{m}(\mathbf{y}_{train}) \\ \mathbf{m}(\mathbf{y}_{test}) \end{bmatrix}, \begin{bmatrix} \mathbf{k}(\mathbf{X}_{train}, \mathbf{X}_{train}) & \mathbf{k}(\mathbf{X}_{train}, \mathbf{X}_{test}) \\ \mathbf{k}(\mathbf{X}_{test}, \mathbf{X}_{train}) & \mathbf{k}(\mathbf{X}_{test}, \mathbf{X}_{test}) \end{bmatrix} \right)$$

The Gaussian properties give:

$$\mathbf{y}_{test} | \mathbf{X}_{test}, \mathbf{X}_{train}, \mathbf{y}_{train} \sim \mathcal{N}(\boldsymbol{\mu}_{y_{test}}, \boldsymbol{\sigma}^2_{y_{test}})$$

Where

$$\begin{aligned} \boldsymbol{\mu}_{y_{test}} &= \mathbf{m}(\mathbf{y}_{test}) + \mathbf{k}(\mathbf{X}_{test}, \mathbf{X}_{train})\mathbf{k}(\mathbf{X}_{train}, \mathbf{X}_{train})^{-1}(\mathbf{y}_{train} - \mathbf{m}(\mathbf{y}_{train})) \\ \boldsymbol{\sigma}^2_{y_{test}} &= \mathbf{k}(\mathbf{X}_{test}, \mathbf{X}_{test}) - \mathbf{k}(\mathbf{X}_{test}, \mathbf{X}_{train})\mathbf{k}(\mathbf{X}_{train}, \mathbf{X}_{train})^{-1}\mathbf{k}(\mathbf{X}_{train}, \mathbf{X}_{test}) \end{aligned}$$

We now understand more easily how GP can be used for regression models. Furthermore, the key to GP modeling is to understand the role of the covariance function. By definition, it links the possible values of the Gaussian process. It can be seen as the *prior* used in Bayesian methods, as it gives information on the distribution we try to model. Thus, by setting the covariance function, we fix the smoothness of the curve.

We also need to notify that the covariance between the *outputs* $\mathbf{y}$ is written as a function of the *inputs* $\mathbf{X}$. For instance, if we consider the zero-mean GP ξ, we can characterize it with its covariance function:

$$cov(\xi(\mathbf{x}), \xi(\mathbf{x}')) = c(\mathbf{x}, \mathbf{x}') = \sigma^2 r(\mathbf{x}, \mathbf{x}')$$

In the latter notation, we introduced r, a parametric correlation function also known as the kernel. It tells how similar $\mathbf{x}$ and $\mathbf{x}'$ are. The most used is called by different names: the radial basis function kernel (RBF), the squared exponential kernel (SE) or the Gaussian kernel, and is defined as follows:

$$r(\mathbf{x}, \mathbf{x}') = exp\left((\mathbf{x} - \mathbf{x}')^T 10^{\boldsymbol{\Omega}}(\mathbf{x} - \mathbf{x}')\right)$$

Here, $\boldsymbol{\Omega}$ is a diagonal matrix with $\boldsymbol{\Omega}_{i,i} = \omega_i$ for $i \in 1, \ldots, d_x$ for d_x the number of features. We now face an issue: r is only defined if $\mathbf{x}$ only takes numerical values. If not, the distance between two values cannot be computed. The problem then is to find the (or at least a) good way to encode categorical variables and finally become able to calculate distances for any mixed variables.

In the case where the Gaussian process is not centered, it is also possible to define a mean function, also parametric, depending on the predictors. We will introduce this in section 2.1 by presenting the LMGP.

1.3 Various mixed variables encoding

1.3.1 The vanilla method: the one-hot encoding

One-hot encoding is an easy-to-understand method, it involves representing categorical variables as binary vectors, where each category is assigned a unique position in the vector. For instance, consider a dataset with a categorical variable "Color" having three categories: red, blue, and green. One-hot encoding transforms this variable into three binary columns, with each column corresponding to a color. Figure 1.3.1 resumes the method with the latter example.

Observation	**Color**
x_1	Red
x_2	Green
x_3	Blue

$\Longrightarrow$ *One-hot Encoding*

Observation	**Red**	**Green**	**Blue**
x_1	1	0	0
x_2	0	1	0
x_3	0	0	1

Figure 1.3.1: Categorical Data and One-hot Encoding

As we can observe, the method creates a new column for each possible value of the categorical variable. If we extend it for multiple features, with various responses for each, the number of columns created would become too important and the data too sparse. This sparsity can be problematic, especially when dealing with large datasets, as it introduces a considerable amount of redundancy and may hinder the learning process.

Furthermore, one-hot encoding consider the distance between each category remains the same, while we could work with a criterion scoring with different values like *Excellent*, *Very Good*, *Good*, *Fair*, *Poor*. In that case, *Very Good* should be closer to *Excellent* than to *Poor*. A way to create adequate distances is to introduce a latent space, in which the features are mapped and distances can be computed. It refers to a space that is not directly observable but is inferred or learned from the available data. LVGPs and LMGPs are built on it, allowing them to capture the underlying structure of the data.

1.3.2 Mapping the variable into different latent spaces: the LVGP method

Latent Variable Gaussian Processes (LVGPs) are a powerful extension of traditional Gaussian processes, specifically designed to address the challenges associated with modeling categorical data. In the context of Gaussian processes, which are widely employed for regression and classification tasks, LVGPs introduce latent variables to enhance the modeling capabilities, particularly when dealing with categorical input features.

In Gaussian processes, the choice of a suitable kernel function is crucial to capture the underlying patterns in the data. However, when categorical variables are present, conventional approaches like one-hot encoding can significantly increase dimensionality,

leading to challenges in model training and increased risk of overfitting. This is where LVGPs come into play, offering a more nuanced and effective solution. By introducing latent variables that represent the unobserved factors influencing the relationship between input features and the target variable, they directly model them, providing a more meaningful representation of the categorical information.

Furthermore, LVGPs can be viewed as a form of Bayesian non-parametric modeling, allowing for flexibility in capturing complex relationships within the data. The mapping of the features becomes a new hyperparameter that the model can optimize to better understand the data and the closeness of some categories. If we consider the latter example with the criterion scoring, LVGPs will more likely understand the ordinal relationship between each level, whereas one-hot encoding was unable to do so.

However, by creating a latent space for each feature, LVGPs fail to capture the full extent of interactions between variables. For instance, if we try to predict the number of ice cream cones sold (y), depending on the temperature (X_1) and humidity (X_2). If we have $X_1 = High$, we might want to expect a high number of sales. However, if $X_1 = High$ and $X_2 = High$, then the weather would most likely be stormy and not suitable for ice cream consumption.

1.3.3 The LMGPs and their single latent space

N. Oune and R. Bostanabad showed that, at least for the data they were working with, a single latent space would be enough to resume most of the underlying relationships between the features [1]. Hence, by using both a prior representation for each combination of categorical variable and a mapping function to the latent space, they finally obtain points of a space in which distances can be computed (by the covariance function), giving information about the similarity between two observations. Furthermore, the *prior* provides information about the expected interaction between variables, enabling the model to better adjust.

2 Implementation of LMGP

2.1 Mathematical Formulation of LMGP

In our project, we employed Latent Map Gaussian Processes (LMGP) to enable the use of categorical variables as predictors. This method is presented in the article by Nicholas Oune and Ramin Bostanabad [2]. The principle is to facilitate the use of Gaussian processes with mixed data (categorical and quantitative) by projecting qualitative variables into a quantitative latent space. From this perspective, the LMGP method is very similar to the LVGP method from the article by Zhang et al. [4]. However, the difference lies in the fact that LMGP uses a single latent space for all qualitative variables, whereas LVGP employs one space per variable. Once the projection is performed, LMGP can fit a classical Gaussian process since all variables have become quantitative. Here are some mathematical details about the LMGP method:

Let $\mathbf{X}_{quant}$ and $\mathbf{X}_{qual}$ be matrices containing quantitative and qualitative variables, respectively, with individuals in rows and variables in columns. It is assumed that variables in $\mathbf{X}_{qual}$ are encoded in "one-hot" as presented in section 1.3.1. Both matrices have n rows and respectively p_{quant} and p_{qual} columns. Let $\mathbf{A}$ be the projection matrix with p_{qual} rows and d columns, where d is the dimension of the latent space into which we want to project qualitative variables. Often, d=2 is chosen. The mean function of the Gaussian process we will fit is given by $\mathbf{F}\boldsymbol{\beta}$, where $\boldsymbol{\beta}$ is a vector of size $h \times 1$, and $\mathbf{F}$ is a matrix of size $n \times h$ defined by $\mathbf{F}_{i,j} = f_j(\mathbf{X}_{quant}[i,.], \mathbf{X}_{qual}[i,.])$, with f_1, ..., f_h being the chosen basis functions to describe the process. Often, $h = 1$ is selected, and f_1 is the constant function equal to 1. This corresponds to fitting a Gaussian process with a constant mean function. To simplify notation, we will refer to the variables associated with individual i, namely the vector $[\mathbf{X}_{quant}[i,.], \mathbf{X}_{qual}[i,.]]$, as $\mathbf{x}_i$. Now, we have all the elements to define a family of kernel functions $r_{\mathbf{A},\omega}$. For all i, j, the kernel function between individuals i and j is given by:

$$r_{\mathbf{A},\boldsymbol{\omega}}(\mathbf{x}_i, \mathbf{x}_j) = r([\mathbf{X}_{quant}[i,.], \mathbf{X}_{qual}[i,.] \times \mathbf{A}] \ ; \ [\mathbf{X}_{quant}[j,.], \mathbf{X}_{qual}[j,.] \times \mathbf{A}] \ ; \ \boldsymbol{\omega})$$

Where $r_{\mathbf{A},\boldsymbol{\omega}}(x_i, x_i) = 1$, $r_{\mathbf{A},\boldsymbol{\omega}}(x_i, x_j) = r_{\mathbf{A},\boldsymbol{\omega}}(x_j, x_i)$, $r_{\mathbf{A},\boldsymbol{\omega}}(x_i, x_j) \in [0, 1]$, and $\boldsymbol{\omega} = (\omega_i)_{i \in [\![1, p_{quant}]\!]}$ is the vector containing the weights of quantitative variables in the calculation of r.

Thus, we have fully defined a family of Gaussian processes on our data, parameterized by $\mathbf{A}$, $\boldsymbol{\omega}$, $\boldsymbol{\beta}$, and σ. The likelihood of the model is then expressed as below, defining the matrix $\mathbf{R}_{\mathbf{A},\boldsymbol{\omega}} = (r_{\mathbf{A},\boldsymbol{\omega}}(x_i, x_j))_{i,j \in [\![1,n]\!]}$:

$$L_{\mathbf{A},\boldsymbol{\omega},\boldsymbol{\beta},\sigma} = \det(2\pi\sigma^2 R_{\mathbf{A},\boldsymbol{\omega}})^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\mathbf{Y} - \mathbf{F}\boldsymbol{\beta})^T (\sigma^2 R_{\mathbf{A},\boldsymbol{\omega}})^{-1} (\mathbf{Y} - \mathbf{F}\boldsymbol{\beta})\right)$$

We aim to find the parameters that maximize this quantity. In practice, it can be shown that :

$$\hat{\boldsymbol{\beta}} = [\mathbf{F}^T \mathbf{R}_{\hat{\mathbf{A}},\hat{\boldsymbol{\omega}}}^{-1} \mathbf{F}]^{-1} \mathbf{F}^T \mathbf{R}_{\hat{\mathbf{A}},\hat{\omega}}^{-1} \mathbf{Y}$$

and

$$\hat{\boldsymbol{\sigma}}^2 = \frac{1}{n} (\mathbf{Y} - \mathbf{F}\hat{\boldsymbol{\beta}})^T \mathbf{R}_{\hat{\mathbf{A}},\hat{\boldsymbol{\omega}}}^{-1} (\mathbf{Y} - \mathbf{F}\hat{\boldsymbol{\beta}})$$

Therefore, it is sufficient to maximize the likelihood with respect to $\boldsymbol{\omega}$ and $\mathbf{A}$ by replacing the other quantities with the relations above.

Because Gaussian processes yield Gaussian vectors on a finite-sized sample (which is always the case in practice), we can use the formulas for conditioning Gaussian distributions to find the expected value and the covariance matrix of new data, conditional on the training data. The article [2] provides the formulas for predicted mean and covariance. If we have new data $x^* = [x^*_{quant},\ x^*_{qual}]$ and denote y^* as the associated response, $f(x^*) = [f_1(x^*), ..., f_h(x^*)]$, and $g(x^*) = [\hat{\sigma}^2 r_{\hat{A},\hat{\omega}}(x_1, x^*), ..., \hat{\sigma}^2 r_{\hat{A},\hat{\omega}}(x_n, x^*)]^T$, the predictive formulas are then given as follows:

$$\hat{\mathbb{E}}[y^*] = f(x^*)\hat{\beta} + g(x^*)^T (\hat{\sigma}^2 R_{\hat{A},\hat{\omega}})^{-1} (Y - F\hat{\beta})$$

$$\begin{aligned} \hat{cov}(y_1^*, y_2^*) = {} & \hat{\sigma}^2 r_{\hat{A},\hat{\omega}}(x_1^*, x_2^*) - g(x_1^*)^T (\hat{\sigma}^2 R_{\hat{A},\hat{\omega}})^{-1} g(x_2^*) \\ & + (f(x_1^*)^T - F^T (\hat{\sigma}^2 R_{\hat{A},\hat{\omega}})^{-1} g(x_1^*))^T \\ & \times (F^T (\hat{\sigma}^2 R_{\hat{A},\hat{\omega}})^{-1} F)(f(x_2^*)^T - F^T (\hat{\sigma}^2 R_{\hat{A},\hat{\omega}})^{-1} g(x_2^*)) \end{aligned}$$

It is also possible to assume that the data is noisy. In this case, it is necessary to slightly modify the kernel function using a regularization parameter called nugget, by setting:

$$r_{new}\mathbf{A}, \boldsymbol{\omega}(\mathbf{x}_i, \mathbf{x}_j) := r_{\mathbf{A},\boldsymbol{\omega}}(\mathbf{x}_i, \mathbf{x}_j) + \delta_{i,j} \times \text{nugget}$$

with

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

In this case, the nugget can be fixed, but it can also be considered as an additional parameter to be optimized by maximum likelihood. Indeed, it is added to the diagonal of the matrix R, on which the likelihood depends. It is important to note that the nugget is added if $i = j$, not if $x_i = x_j$. Indeed, this parameter concerns the intrinsic variance of the observations rather than the covariance between two observations having the same predictors.

The documentation for our LMGP program is available in the appendix.

2.2 Fitting LMGP Models

In this section, we demonstrate the fitting of our LMGP on simulated data. We discuss fitting with and without noise in the data, as well as the representation of latent spaces in the two-dimensional case.

2.3 Simple Examples

We begin with a simple example. We fit our model to the following data: We have two predictors (x and bin). x takes values in the interval [0,1], and bin is a binomial variable taking either 0 or 1 with a probability of 0.5 each. The observations are generated as follows: $y = \cos(10x) \times \text{bin} + 4(\text{bin} - 1) + x \times \text{bin} + x\sin(x)$. We fitted an LMGP with a Gaussian kernel on a sample of 60 training points.

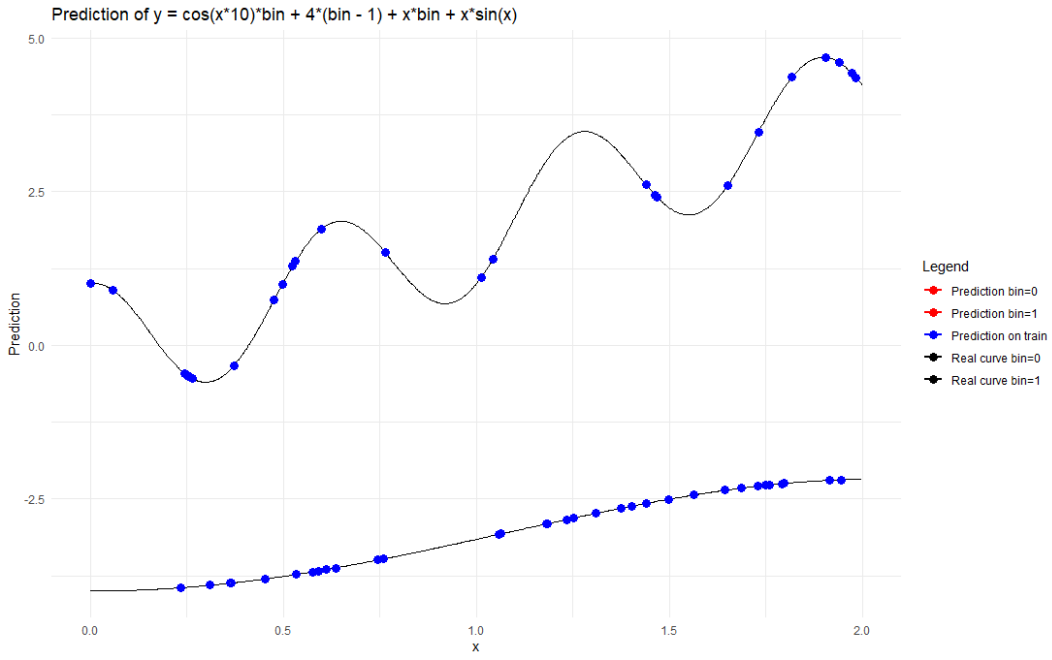

Figure 2.3.1: Prediction of a function depending on a categorical variable without noise

In the above graph, the red line representing the model predictions is invisible as it perfectly coincides with the actual curve. Thus, we have a perfect prediction. We can observe that the LMGP has learned to predict the curves for each of the two levels of the bin variable, although these curves are not of the same shape. Indeed, the upper curve, corresponding to bin = 1, is sinusoidal, while the lower curve is not. The graph also displays the 95% confidence interval, but it is invisible as it is very narrow around the predictions. The absence of noise in the data allows the model to fit with very little uncertainty.

Now let's see what happens when adding slight noise to the data:

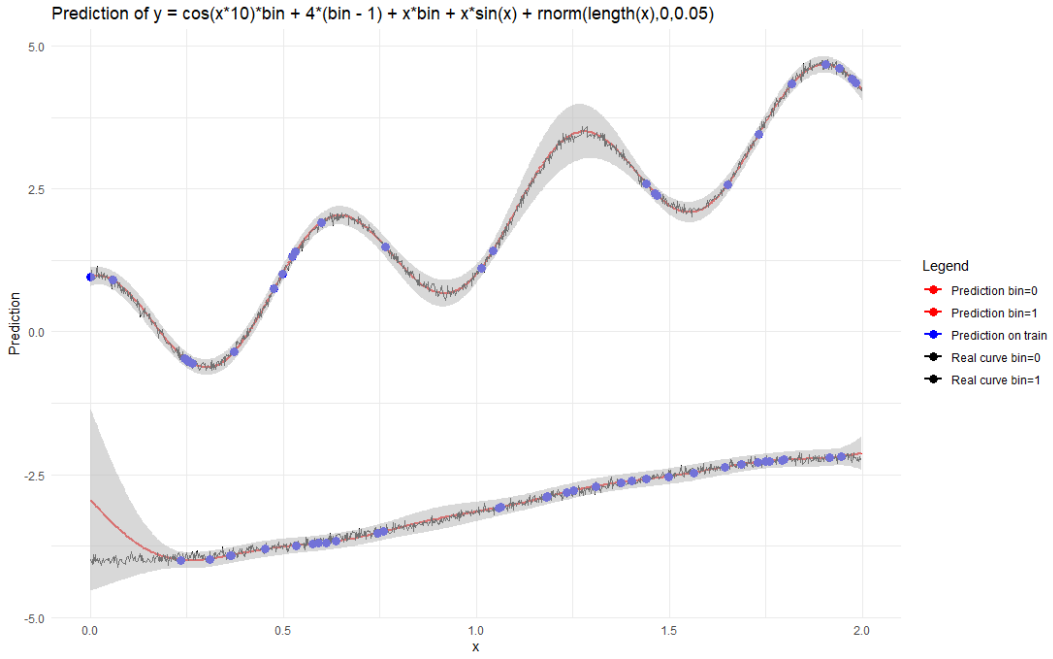

Figure 2.3.2: Prediction of a function depending on a categorical variable with noise

With some noise, the LMGP prediction remains good. Indeed, the red curve closely follows the black curve. However, we have wide confidence intervals, unlike the case without noise. Indeed, the LMGP has adjusted a non-zero regularization parameter in this case. We did not change the model's parameters between the first and second cases. This means that the LMGP has autonomously identified the presence of noise and adjusted its regularization parameter accordingly. When we are close to a training point (in blue), the uncertainty is roughly equal to the noise uncertainty in the data. However, when we are between two training points, the uncertainty is larger because there is also doubt about the model fit, in addition to the data noise.

2.4 Examples from the article and latent spaces

We also tested our implementation on the functions mentioned in the article [2]. Here 2.4.1, we show an example of model fitting on the Borehole function. The qualitative variables are projected into a quantitative latent space, the dimension of which we can choose. In order to visualize the results and limit the number of parameters to optimize, we chose a latent space of dimension 2:

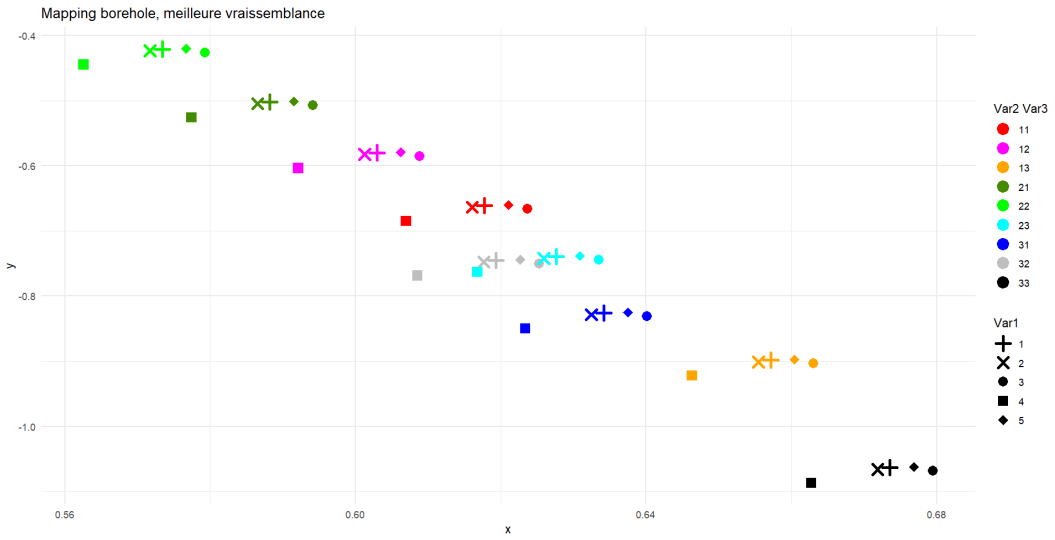

Figure 2.4.1: Latent space, Borehole function, 100 samples, MSE = 5.08

This represents the projection of every combination of every level of the categorical data from the borehole function in a learned latent space. Shapes represent each level of the first level of the first categorical variable, color represents a similar combination of level for the second and third. For each color, the same pattern is repeated with a translation. Also, the shapes are aligned. This reflects the fact that for each color, the ratio var2/var3 is constant, and similar shapes have the same level of var1. The projections in the latent space reflects certain properties of the function they come from. This property is shown in the paper at an ever greater scale, *cf* figure 2.4.2

Although we were unable to precisely reproduce the latent space of the Borehole function presented in the graph, we can see from Figure 2.4.2, associated with the highest likelihood we found, that lines and a parallelogram shape are recovered. Here is the original graph from the article. We are not aware of its conditions (number of training points, multistart, optimization method, etc.), which made its reproduction challenging.

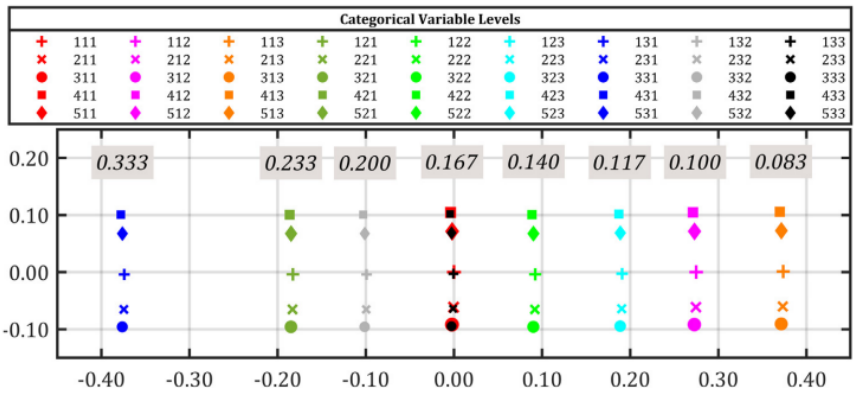

Figure 2.4.2: Original Borehole latent space, Source: N. Oune and R. Bostanabad

The projection of qualitative variables is performed via the matrix A presented in Section 2.1, of size (total number of modalities) $\times$ (desired latent space dimension). Therefore, it is a complex optimization problem in high dimension.

3 Benchmark of LMGP's predictive power

3.1 Benchmark methodology

In this section, we try to reproduce practical results displayed in the paper, which illustrate the performance of both the LMGP and the LVGP methods in multiple contexts. We chose to reproduce the experiment with three functions: the OLT Circuit function, the Borehole function, and the custom function, all available in the annex (1, 2, 3).These three functions have different levels of non-linearity and of parameters to estimate. After describing the parameters of each function, we will present the results obtained and compared them with the ones from the paper.

OLT Circuit

This function outputs a real value, and takes 6 parameters, 3 of which have been discretized to mimic categorical variables - *cf* in figure 1.

Variable	Type	Range
R_{b1}	Categorical	[1, 3]
R_{b2}	Numerical	[50, 70]
R_f	Categorical	[1, 3]
R_{c1}	Numerical	[1.2, 2.5]
R_{c2}	Categorical	[0.01, 5]
β	Numerical	[1, 3]

Table 1: OLT Circuit Variables

To compute the output of the function, we used real values for the discretized variables, as shown in table 2:

Categorical variable levels	R_{b1}	R_f	β
1	25	0.5	1
2	32.5	2	4
3	40	3	5

Table 2: Categorical variables' underlying values

Borehole

The borehole function is used in the paper to illustrate the power of LMGPs' embeddings. To be sure our implementation of the LMGP is correct, we plotted the projection learnt by the model and compared it with the paper. Borehole was thus a mandatory function to test.

Variable	Type	Range
T_u	Numerical	[100, 1000]
H_u	Numerical	[990, 1110]
H_l	Numerical	[700, 820]
r	Numerical	[100, 10000]
r_w	Numerical	[0.05, 0.15]
T_l	Categorical	[1, 5]
L	Categorical	[1, 3]
K_w	Categorical	[1, 3]

Table 3: Borehole variables

All the variables were initially numerical, but in order to plot the latent positions, the authors decided to convert T_l, L and K_w into categorical variables, described hereafter : 4.

Categorical variable levels	T_l	L	K_w
1	10	1000	6000
2	30	1400	10000
3	100	2000	12000
4	200		
5	500		

Table 4: Categorical variables' underlying values

Custom function

OLT Circuit and Borehole are two main functions used in the paper to compare LMGP and LVGP, but we also wanted to train the models on function with a bit more level combinations. We knew that too many combinations would lead to convergence issues, and we did not have enough time to train them efficiently. Hence, we decided to use the custom function as a good compromise. Hereafter is the description of the variables.

Variables	Type	Range
x_1	Numerical	[0, 1]
x_2	Numerical	[0, 1]
x_3	Categorical	[1,6]
x_4	Categorical	[1,4]
x_5	Numerical	[0, 1]
x_6	Numerical	[0, 1]
x_7	Numerical	[0, 1]
x_8	Categorical	[1,3]

Figure 3.1.1: Custom function Variables

To compute the output of the function, we used real values for the discretized variables, as shown in table 5:

Categorical variable levels	x_3	x_4	x_8
1	0	0	0
2	0.1	0.2	0.4
3	0.3	0.7	1
4	0.6	1	
5	0.7		
6	1		

Table 5: Categorical variables' underlying values

To measure the performance of the models, we used the same methodology as explained in the paper : 10 models fitted on fixed variables, with different levels of noise added to the output. Due to the high calculation time of the LVGP implementation, only LMGP and RF MSE could be benchmarked, the LVGP methods result will be taken from the LMGP paper directly. We considered training samples of size 50, 100, 200, 300 and 400, the error was computed from test samples of size 10 000.

The training data was generated via Sobol sequence within the range of the variables. This sequence was then scaled to match the range of the variables used in each function. The response to those training points was then computed, and Gaussian noise was added to introduce randomness in the training data. Then, models were trained on each sample.

To estimate predictive power, we simulated 10 000 training points by using uniform distributions within each variable's range of values. We then computed the response of the testing points and added noise to the outputs - the same noise used for the training set. Finally, we predicted those testing outputs with the models before computing the Mean Squared Error (MSE) on the results. This procedure was repeated 10 times per combination of training sample and noise variance, to account for randomness. These repetitions allowed us to present the results with boxplots, giving hints on the variability of each model's performances.

The optimization was done using the L-BFGS-B algorithm with the optim function of R, using Latin Hypercube Sequences for multistarts.

Three multistarts strategies were tested, Sobol sequences, Latin Hypercube Sequence - LHS, and uniform random initialization. The latter seemed to perform worse than the other two in our tests, we chose to use LHS for our benchmark to avoid using the same starts at each iteration - some results seemed biased with Sobol initialization.

3.2 Benchmark Results: Similar results to the paper, with hints of likelihood optimization issues

In this section, we compare the performance of our implementation to the performance presented in the paper. As references, we chose to use Random Forests - RF thereafter, and the LVGP results from the paper - due to high calculation time, we could not include this model in our benchmark.

It should be noted that the uniform multistart will not be shown here, due to worse performance in our testing. First we will showcase the OLT circuit benchmark, with 12 multistarts using Sobol sequence 3.2.2.

The results from the paper are presented in figure 3.2.1,

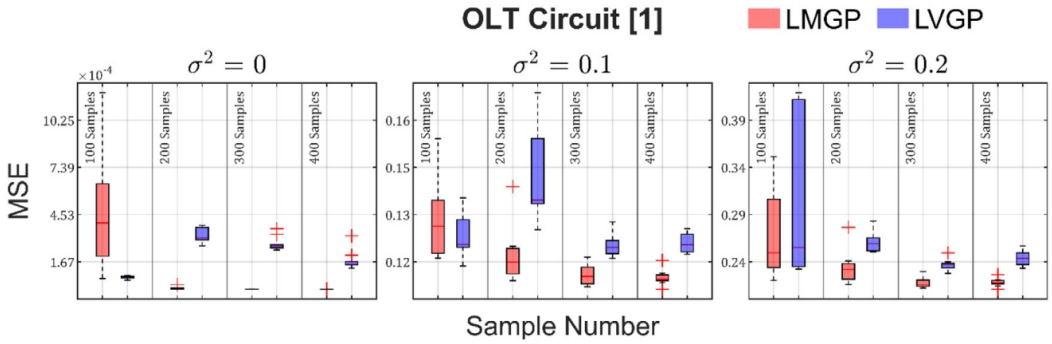

Figure 3.2.1: Results of benchmark between LMGP and LVGP for the OLT circuit function, provided in the studied paper

Source: Latent map Gaussian processes for mixed variable metamodeling, N. Oune and R. Bostanabad

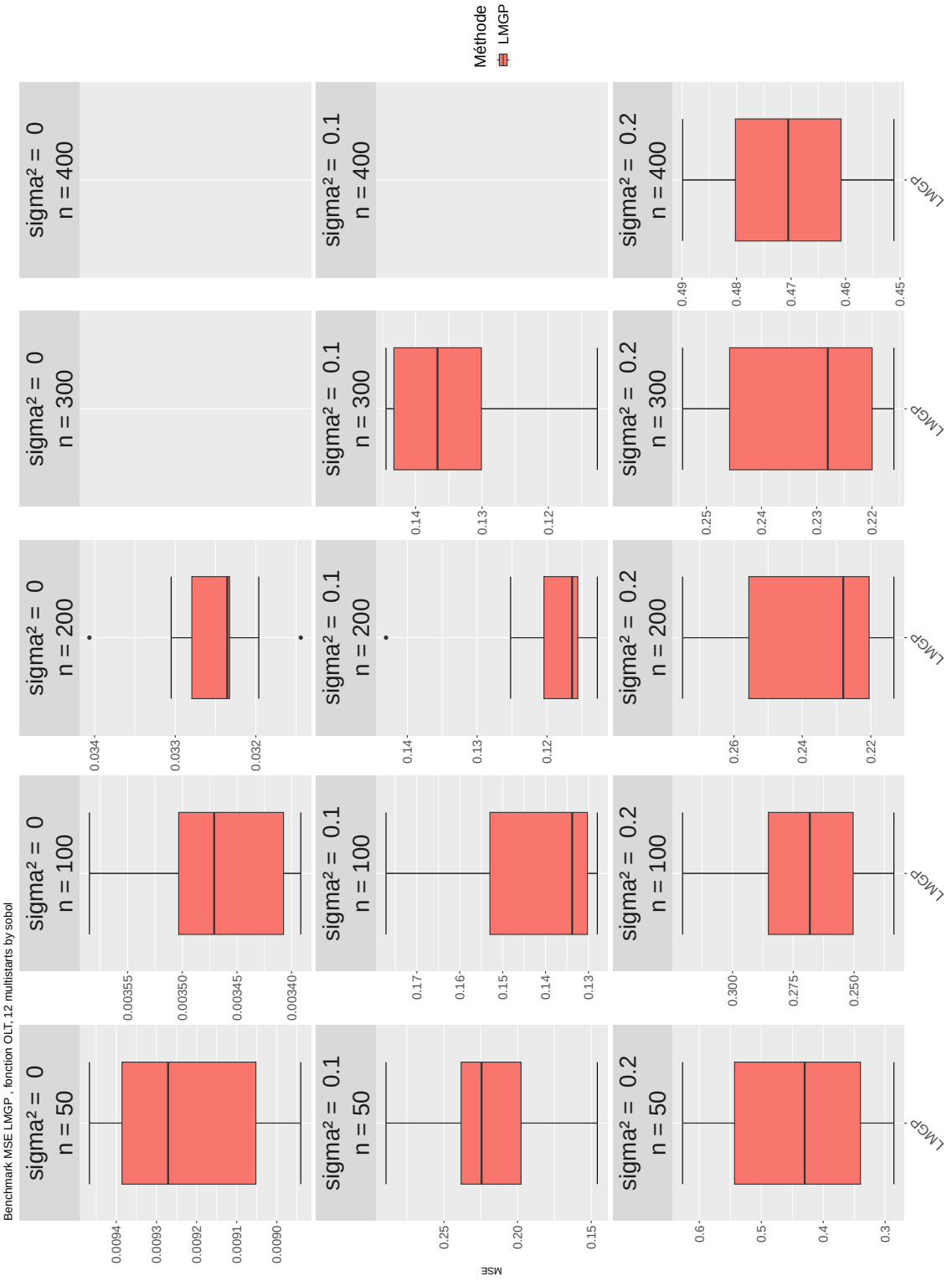

Figure 3.2.2: Results of LMGP benchmark on OLT circuit function, simulated data, 12 multistarts with Sobol sequence initialization

Comparing our results to the performance showcased in the paper, we have similar order of MSE on all level with a training sample size of 100, but our implementation is slightly worse with higher sample size.

Note that with 300 and 400, for cases with respectively 0,and 0 or 0.1 noise variance, none of the models converged. This could be due to the initializations, that were the same amongst all repetitions due to the nature of Sobol sequences. This is mainly why we chose to use LHS in our other benchmarks, that is intermediate between uniform random and Sobol sequences based multistarts. Another downside of the Sobol sequence initialization strategy is that all models share the same initializations. This lead to biased measures of performances in our tests, with examples where all repetitions converged by what we assumed to be luck.

Lastly, our results pinpoints other convergence issues. In facts, the more the data used for training, the worse the MSE was. This could lead to a need for a greater number of initializations as the training sample size grows. This is why we chose to use 50 multistarts thereafter.

With all these changes, we obtain figure 3.2.3. This figure shows that by increasing the number of initializations and using LHS, we obtain better convergence with low variance cases. On average, he levels of MSE are rather similar than with 12 initializations, but some models reached lower MSE.

For the borehole and the custom function, figures 3.2.4 and 3.2.7, similar trends can be observed, with higher intensity, *c.f.* respectively 3.2.5 and 3.2.6 for reference. This could come from the higher dimension of the latent space to estimate, due to a higher number of level for each categorical variables. This leads to more parameters to optimize while learning the mapping matrix.

The same figures with RF performance as comparison are available in annex 6.0.1, 6.0.2 and 6.0.3. Training times are available in annex 6.0.5, 6.0.5 and 6.0.5

This gap between our implementation and the results of the paper [2] could mainly come from our optimization of the pseudo-likelihood. We gathered multiple hints of lack of convergence in our optimization routines. Even with multiple initialization strategies - uniform random start, Sobol sequence start or LHS, our algorithms seemed to reach local minimums only. Further evidence will be brought in the following part.

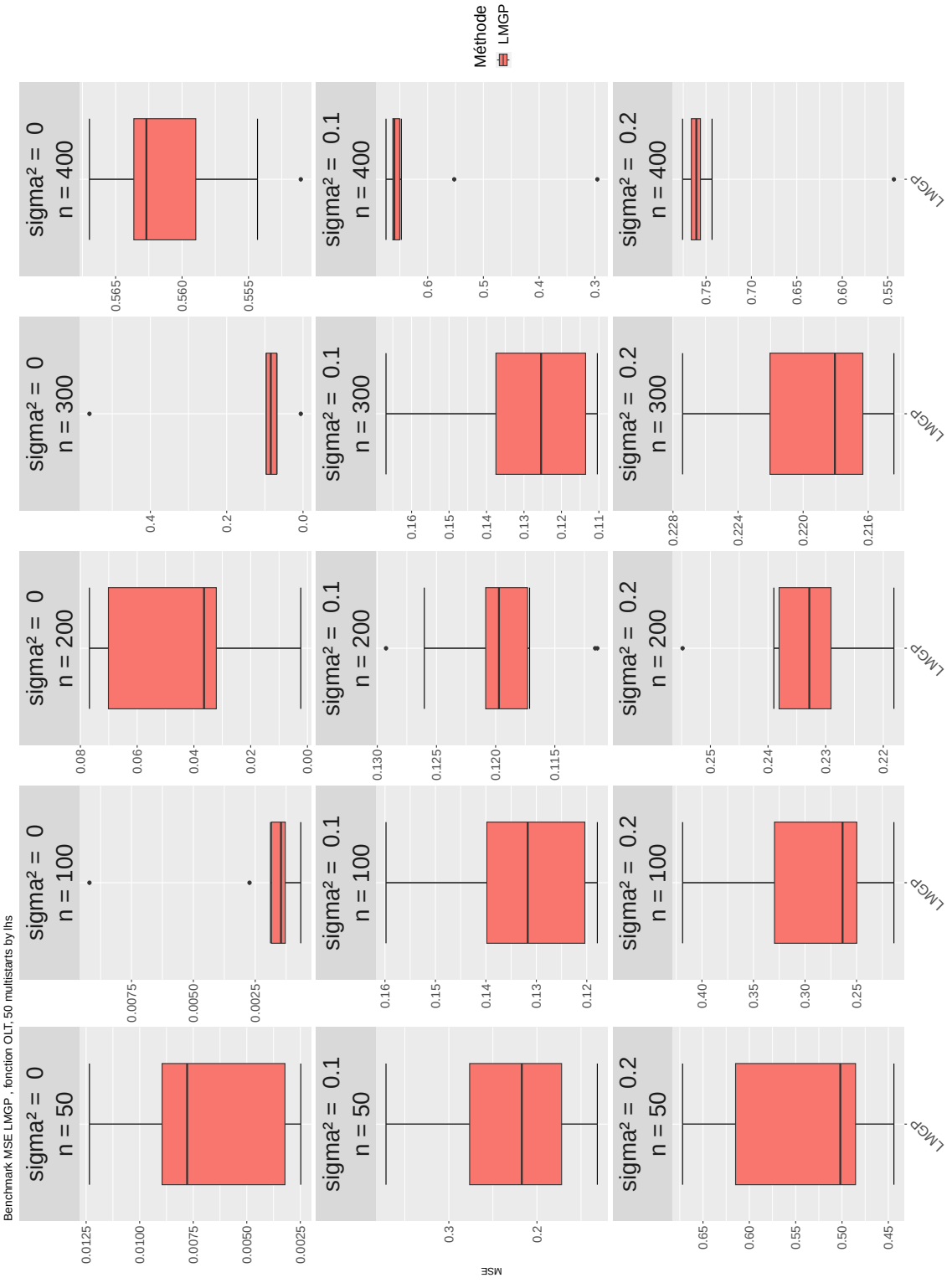

Figure 3.2.3: Results of LMGP benchmark on the OLT circuit function, simulated data, 50 multistarts with LHS initialization

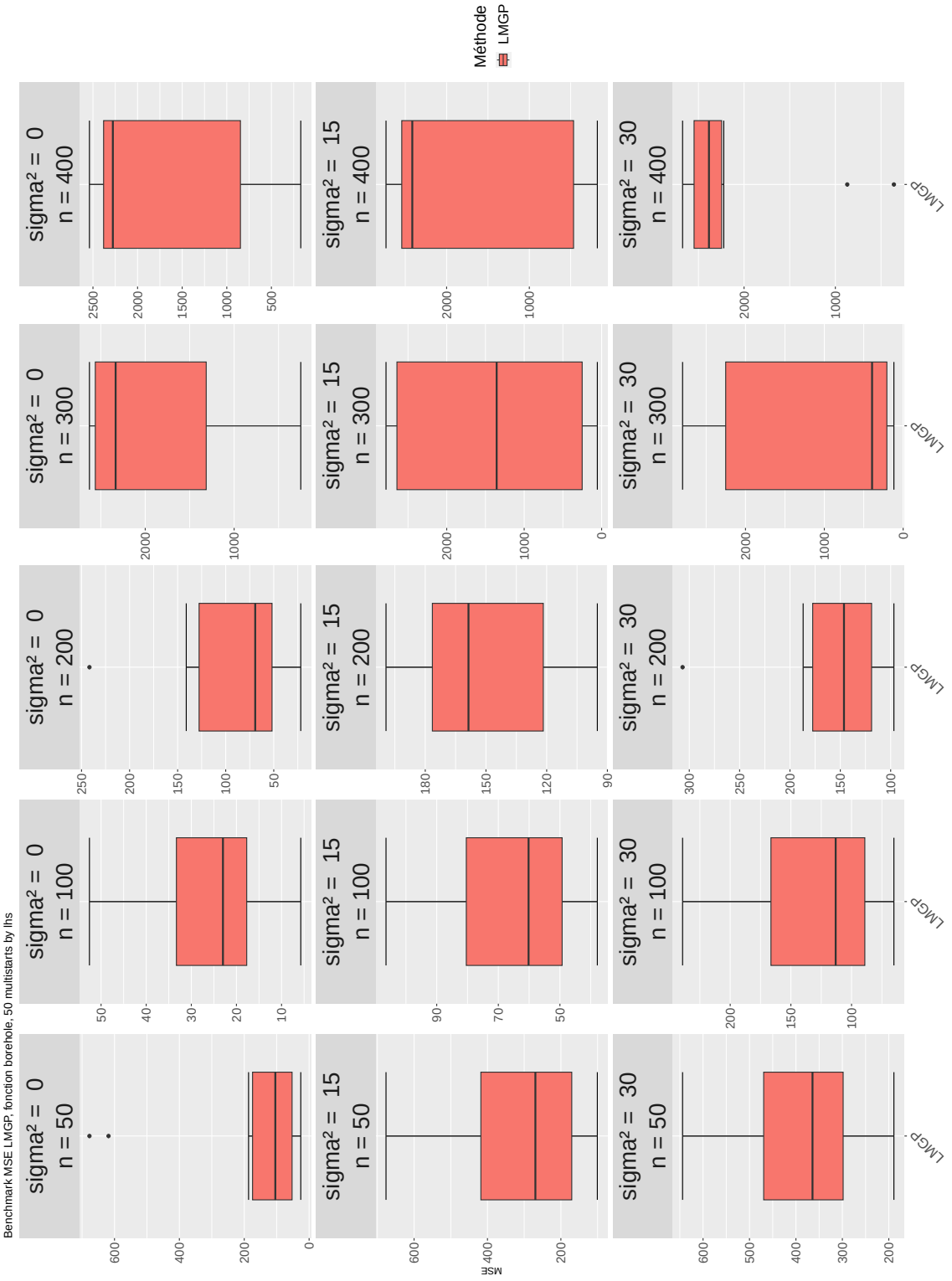

Figure 3.2.4: Results of LMGP benchmark on the borehole function, simulated data, 50 multistarts with LHS initialization

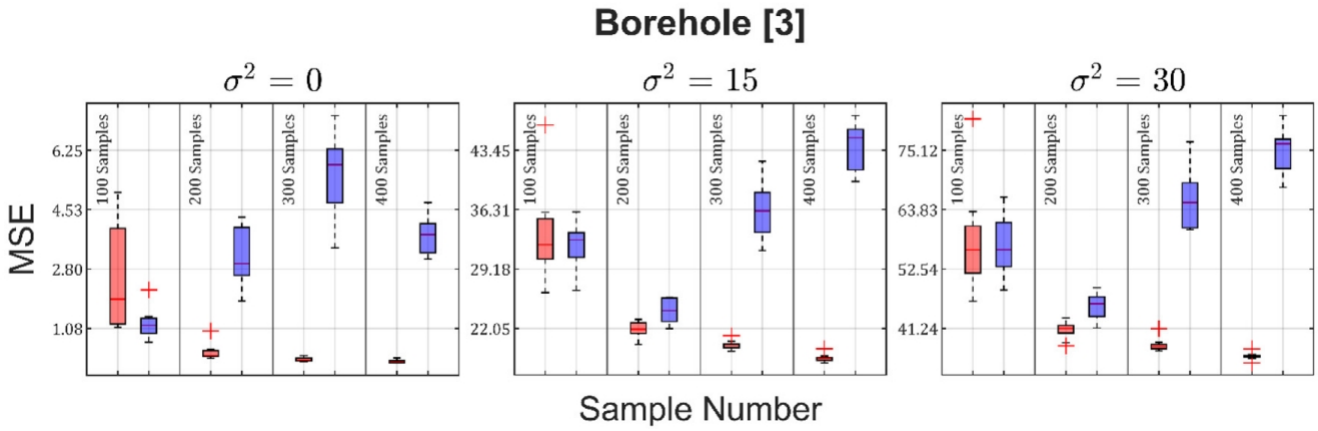

Figure 3.2.5: Results of benchmark between LMGP and LVGP for the borehole function, provided in the studied paper

Source: Latent map Gaussian processes for mixed variable metamodeling, N. Oune and R. Bostanabad

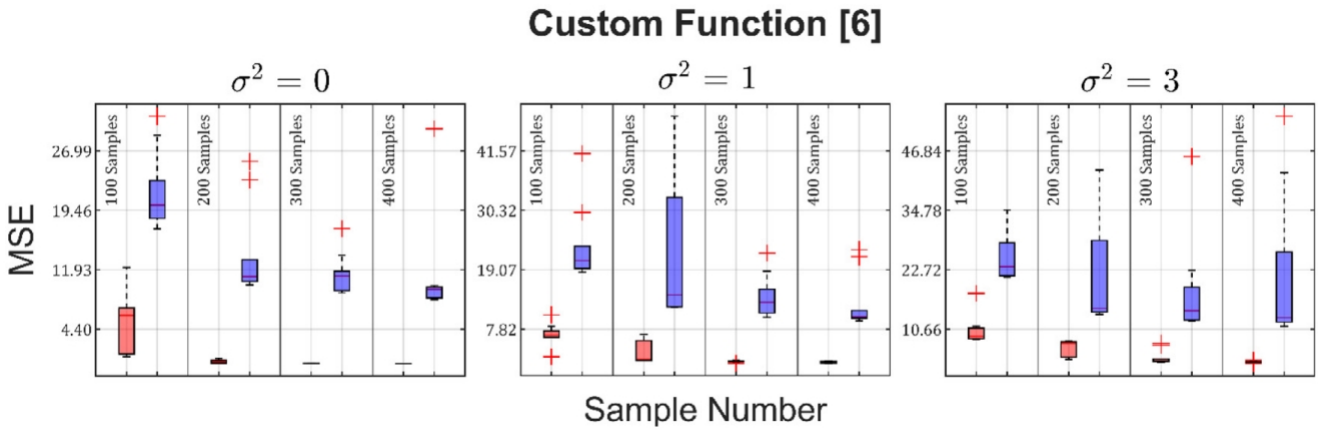

Fig. 7. (*continued*).

Figure 3.2.6: Results of benchmark between LMGP and LVGP for the custom function, provided in the studied paper

Source: Latent map Gaussian processes for mixed variable metamodeling, N. Oune and R. Bostanabad

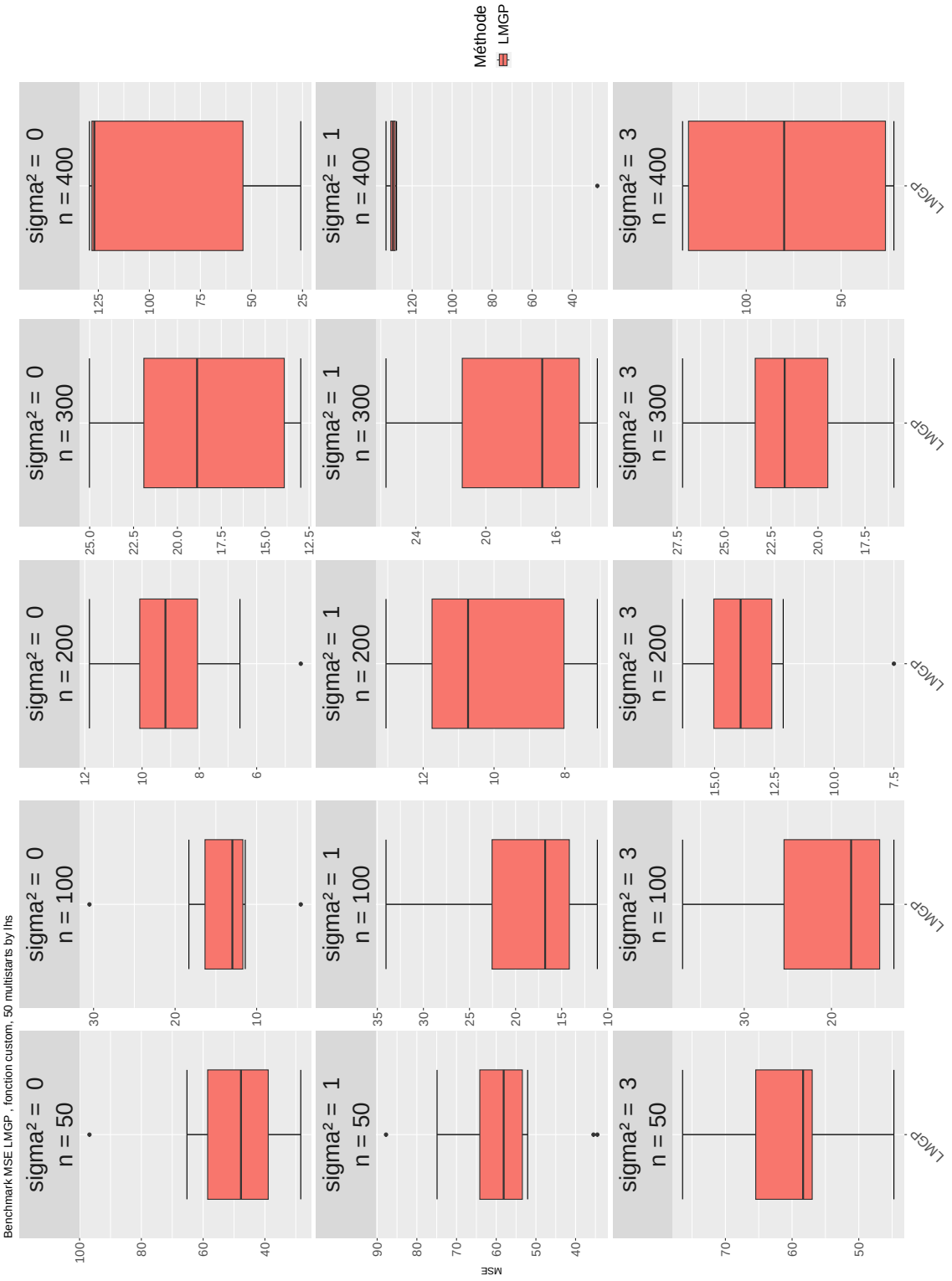

Figure 3.2.7: Results of LMGP benchmark on the custom function, simulated data, 50 multistarts with LHS initialization

3.3 Comparing Gaussian and Matern kernels

We provide a brief comparison of the performance of the Gaussian kernel against the Matern $\frac{5}{2}$ kernel. We tested these two kernels on the OLT circuit and Borehole functions, with 100 observations and a multistart parameter set to 100. We repeated the experiment 200 times. The data were noised with a variance of 30 for Borehole and 0.2 for OLT circuit. Here are the results obtained:

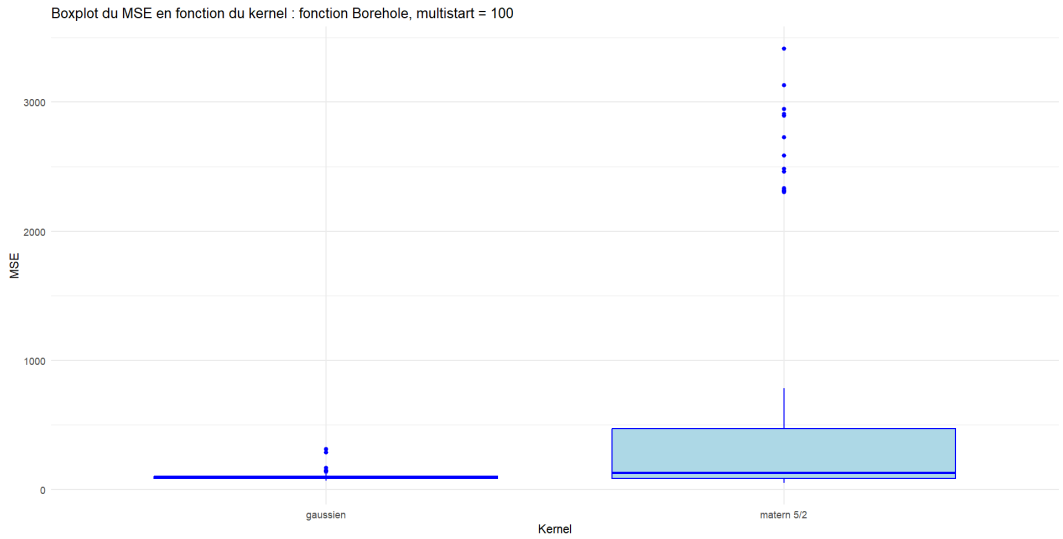

Figure 3.3.1: Gaussian kernel vs Matern kernel, Borehole, $\sigma^2 = 30$

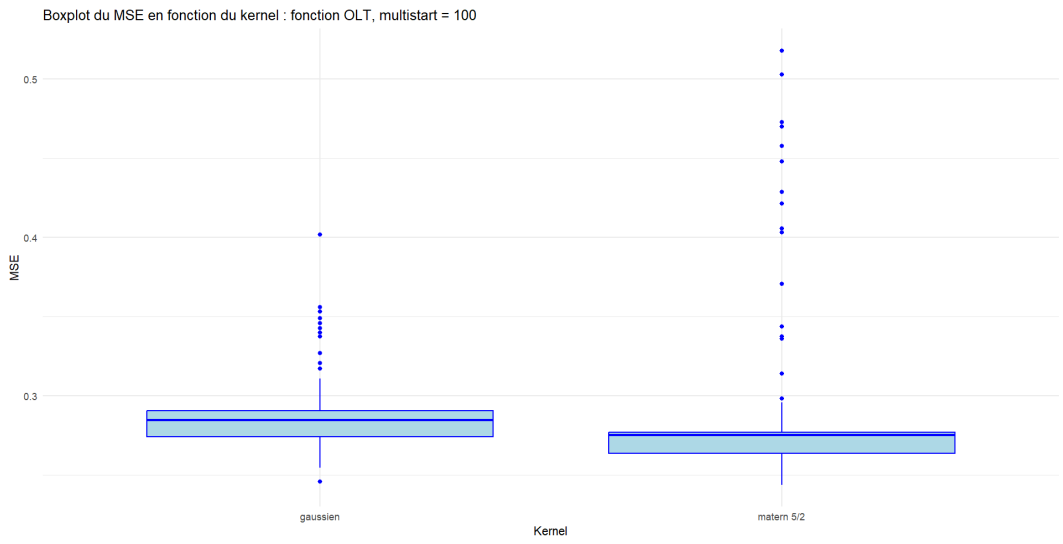

Figure 3.3.2: Gaussian kernel vs Matern kernel, OLT circuit, $\sigma^2 = 0.2$

We can see that the Gaussian kernel seems better on the Borehole data, while the Matern kernel appears to perform better on the OLT circuit data. We also observe that

convergence issues seem more frequent with the Matern kernel than with the Gaussian kernel. We will have the opportunity to revisit these optimization problems later, as fitting LMGP models is a complex optimization problem in high dimensions. We also note that despite the multistart, many models have very large MSEs. However, we are reassured to see that the best models indeed achieve the MSEs presented in the article [2].

4 Benchmark on a real dataset

4.1 Description of the data and explanation of the methodology

To evaluate the effectiveness of our LMGP implementation, we conducted a comparative analysis using real-world datasets. We measured its performance against both LVGP and RandomForest algorithms. This gave us a good sense of how well LMGP holds up in the real world compared to these other methods.

Structure of the data

To test the LMGP efficiency, we used a dataset with **13 variables** and **223 observations**. The predictor variables are the first ten, and the outcome, a continuous variable, is the twelfth. Below is a description of the dataset.

Variable	Type	Range / number of categories
X_1	Categorical	[1, 10]
X_2	Categorical	[1, 12]
X_3	Categorical	[1, 2]
X_4	Numerical	[0.416, 0.617]
X_5	Numerical	[0.4265, 0.829]
X_6	Numerical	[1.36, 1.593]
X_7	Numerical	[0.445, 1.093]
X_8	Numerical	[0.808, 1.382]
X_9	Numerical	[0.521, 0.62]
X_{10}	Numerical	[0.4875, 0.596]
Y	Numerical	[16, 154]

Figure 4.1.1: Description of the real dataset

Methodology

We chose to divide our dataset into a training set consisting of 200 randomly chosen observations and a testing set containing the 23 last observations. This partitioning strategy was motivated by the need to strike a balance between ensuring an adequate amount of data for model training while still retaining a sizable portion for rigorous testing and evaluation. By repeating this operation 50 times, we aimed to account for the variability in performance that may arise due to different random splits of the data. This iterative approach allowed us to obtain more reliable and robust estimates of model performance by averaging the results across multiple trials. Importantly, each model (LMGP, LVGP, and Random Forest) was trained on the same training set to ensure fair and consistent comparison of their respective performances on the testing data. This standardized training protocol facilitated a reliable assessment of the relative strengths and weaknesses of each model under consideration.

In our experimentation, we focused our testing primarily on the performance of LMGP and Random Forest algorithms due to practical constraints regarding computational resources and time efficiency. While we originally intended to include LVGP in our comparative analysis, we encountered challenges during the fitting process. We thus had to use the results provided by Mr. Da Veiga for the performance evaluation of LVGP.

4.2 LMGP outperforms Random Forest, but both are far away from LVGP

First of all, we compare the RMSE calculated on the predictions of LMGP and Random Forest (RF). Upon examination, LMGP demonstrates a slight edge in predictive accuracy over RF. However, none of our results can challenge those obtained using LVGP. Our methods' RMSE vary from 10 to 25, whereas LVGP score is constrained between 0 and 1, testifying a great accuracy. From the explanations of the paper, LMGP should have shown a better accuracy, but the great number of level combinations (here 240) gave our optimizer a hard time. Furthermore, we used only two dimensions for our projection space, limitting the possibility to find the optimal solution.

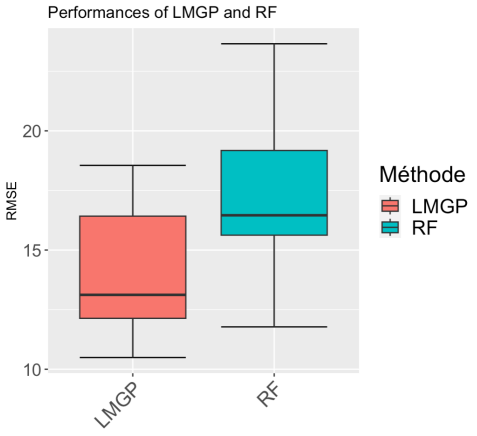

Figure 4.2.1: Comparison of LMGP and RF performances

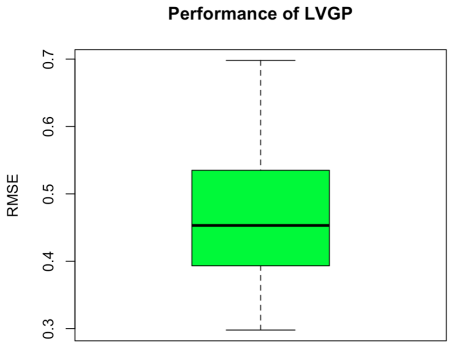

Figure 4.2.2: LVGP performances

On the other hand, the fitting time was excessively long for LVGP, whereas it was almost instantaneous for RF. We are convinced that, with more time, we could have better trained our model to get results closer to the ones of the LVGP.

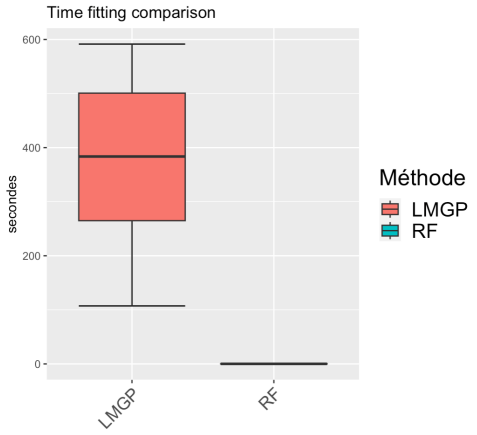

Figure 4.2.3: Comparison of LMGP and RF fitting time

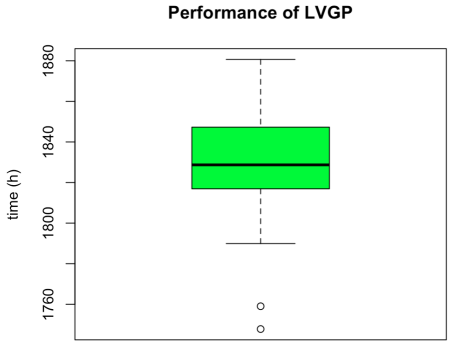

Figure 4.2.4: LVGP fitting time

5 Identification and Attempts to Solve Optimization Problems

5.1 Estimation of MSE Distribution with Randomly Initialized Optimization

Adjusting LMGP has proven to be difficult as the complexity of functions increased. Indeed, a greater number of qualitative variable modes increased the number of parameters through matrix A. Additionally, we identified that the model struggled due to the significant number of local minima. As previously discussed, there are several possibilities for learning a latent space. We have demonstrated that it often occurred that the model learned two different but rotationally identical latent spaces. This is an initial illustration of the local minima we encountered. The following figures also show the quantity of different likelihood maxima found by the model:

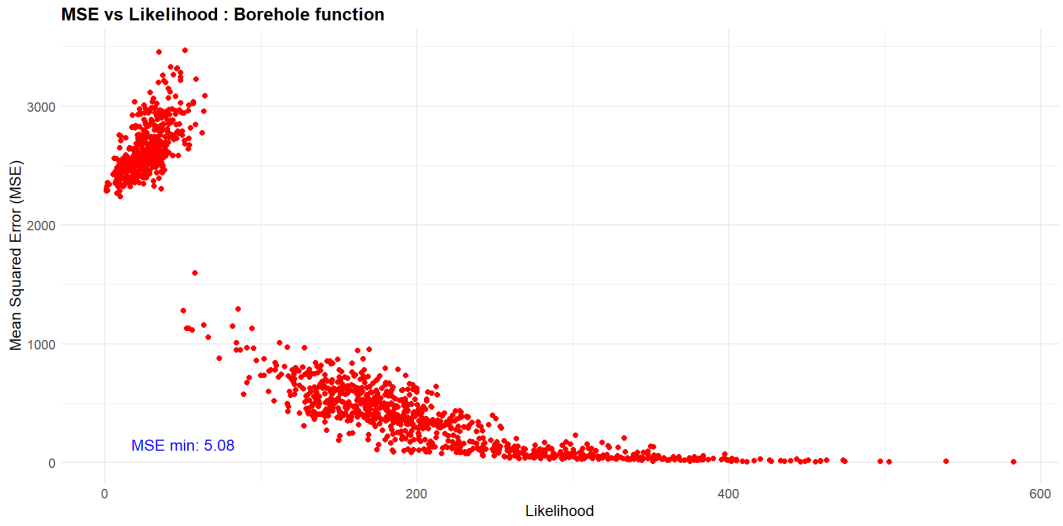

Figure 5.1.1: MSE vs Likelihood, Borehole function, 100 samples

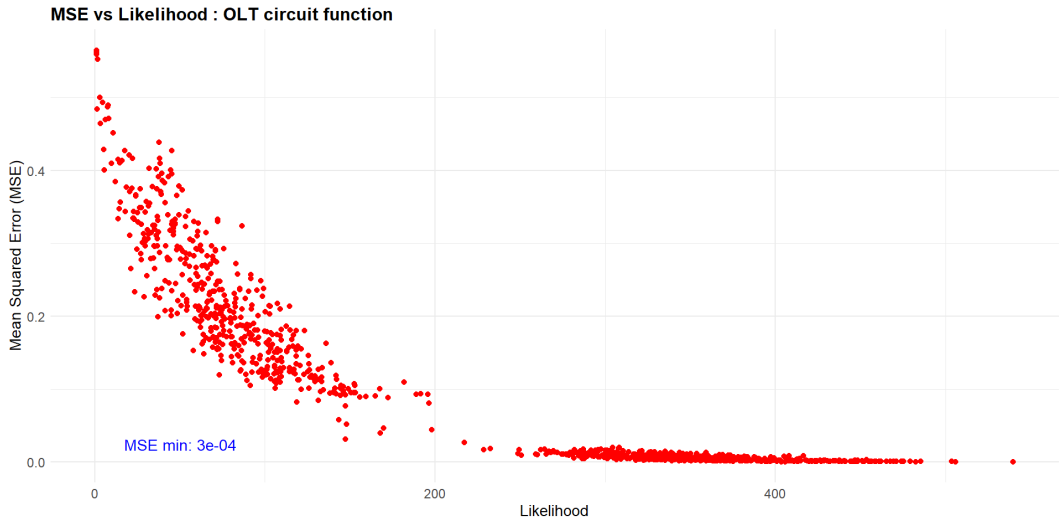

Figure 5.1.2: MSE vs Likelihood, OLT function, 100 samples

We observe very rarely that some models are optimized with a high pseudo-likelihood and a very low MSE, with levels similar or even better as in the paper's benchmark. This is a hint of an absence of errors in our implementation. Indeed, we can see that maximizing likelihood is, with minor variations, similar to minimizing MSE. We obtained these plots by training models with 2000 random initializations.

We can see that passed a certain point in likelihood, many models have similar low MSE. We interpreted this as evidence that similar local likelihood maxima were found. The minimal MSE found here aligns with the levels presented in the article [2]. With higher sample size, models with higher likelihood are fewer, and overall convergence of the optimizer is also less frequent, cf 5.1.2 and 5.1.3.

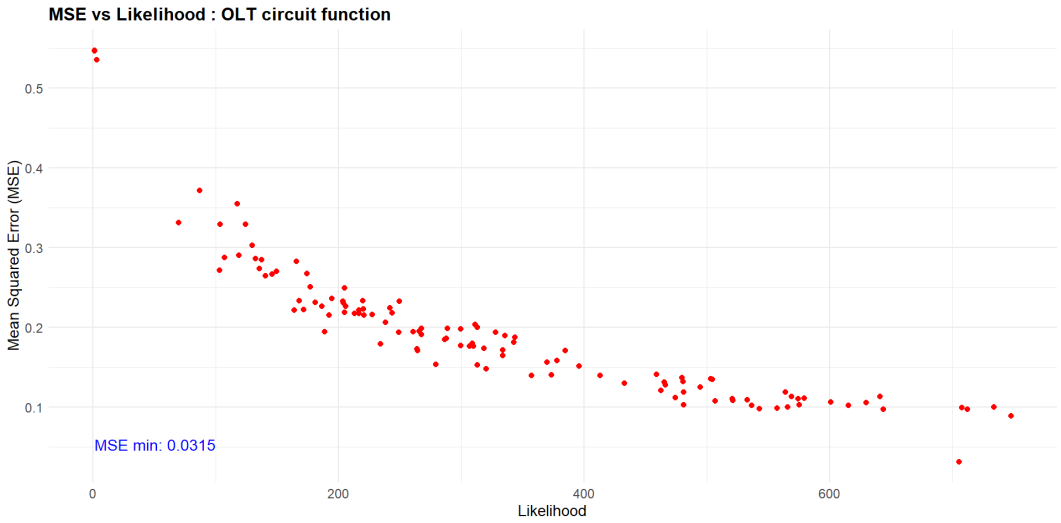

Figure 5.1.3: MSE vs Likelihood, OLT function, 500 samples, var = 0.2

We observe that higher training size requires more random starts to allow convergence of the optimization algorithms, as shown in figures 5.1.4 and 5.1.5. This could explain the degradation of the MSE in the benchmark as the training sample size grew larger.

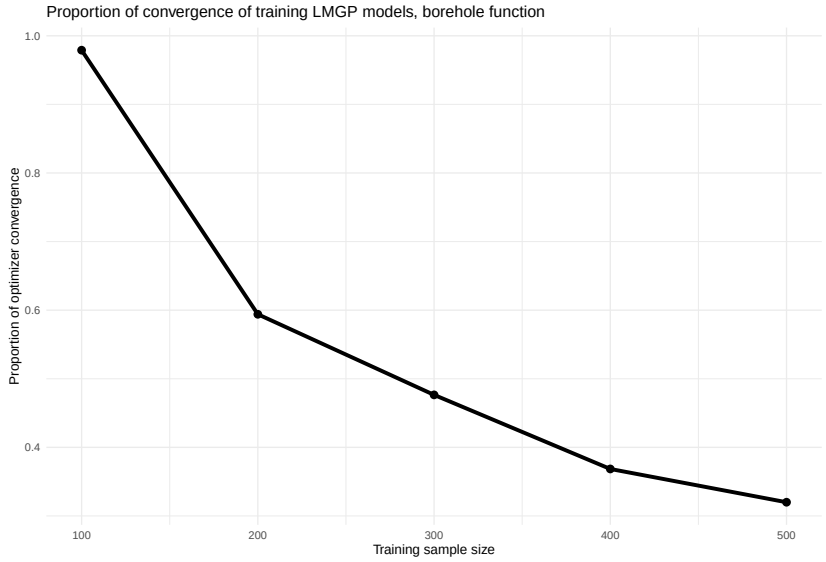

Figure 5.1.4: Proportion of convergence over the training of 2000 models with one random uniform start, borehole function

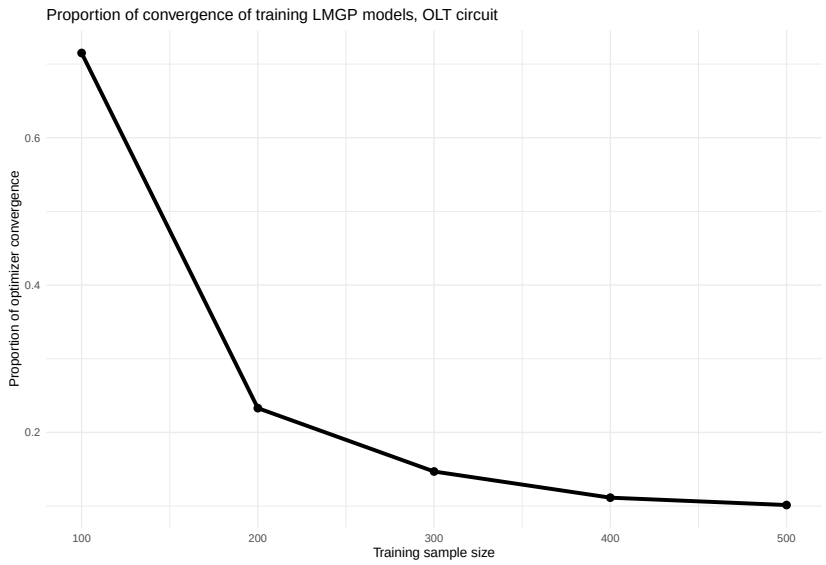

Figure 5.1.5: Proportion of convergence over the training of 2000 models with one random uniform start, OLT circuit function

In summary, with a few models that have similar performance as showcased in the article, we can entrust our implementation and put our optimization method in doubt. With our current methods, 12 initialization, as mentioned in the paper, is far from sufficient to achieve better models. The optimization, with the presence of local optimum, appears to be the explanation to the poor results of our benchmark. In the next section, we propose an explanation to visualize the potential origin of a portion of these local maxima.

5.2 Invariance by rotation : a possible source of local optimum

Another issue encountered within our implementation was to deal with local optimum. The authors proposed a solution to make the optimal solution of the problem invariant by translation and by rotation. If we focus on the result of the prior section, we can see that a few models have a high pseudo-likelihood and a low MSE. Those are considered to be the best models, but if we plot the representation of each combination of each categorical level, we obtain very similar latent spaces, that are nearly identical up to rotation.

For instance, figures 5.2.1 and 5.2.2 show the latent space of respectively the lower MSE and the higher pseudo-likelihood models. We can see that the mapping is the same up to a reflection. The constraints enforcing invariance by rotation and translation could tend to limit the abundance of local optimum by limiting those cases. However, The implementation of those constraints is heavy, and the optimizer became really slow during our tests, which lead us to experiment with other solutions to try to fix our

optimization issues, such as the cmaes optimizer. However, this new method lead to poor results, with even higher MSE with much higher computation time.

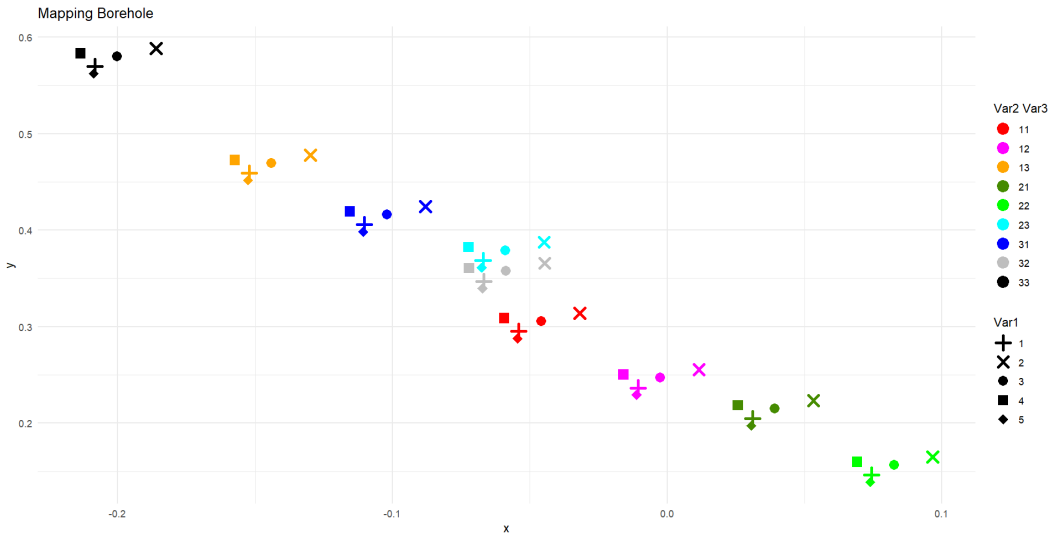

Figure 5.2.1: Latent space, Borehole function, 100 samples, MSE = 5.78

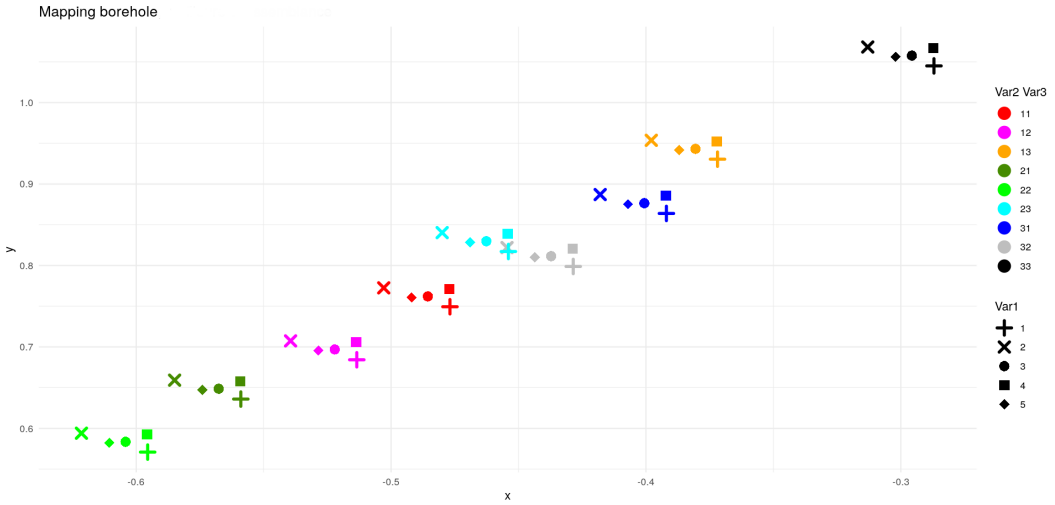

Figure 5.2.2: Latent space, Borehole function, 100 samples, MSE = 6.20

We also observe this phenomenon for the OLT circuit function, as demonstrated by figures 5.2.4 and 5.2.3, which correspond respectively to the model with the best likelihood and the one with the best MSE.

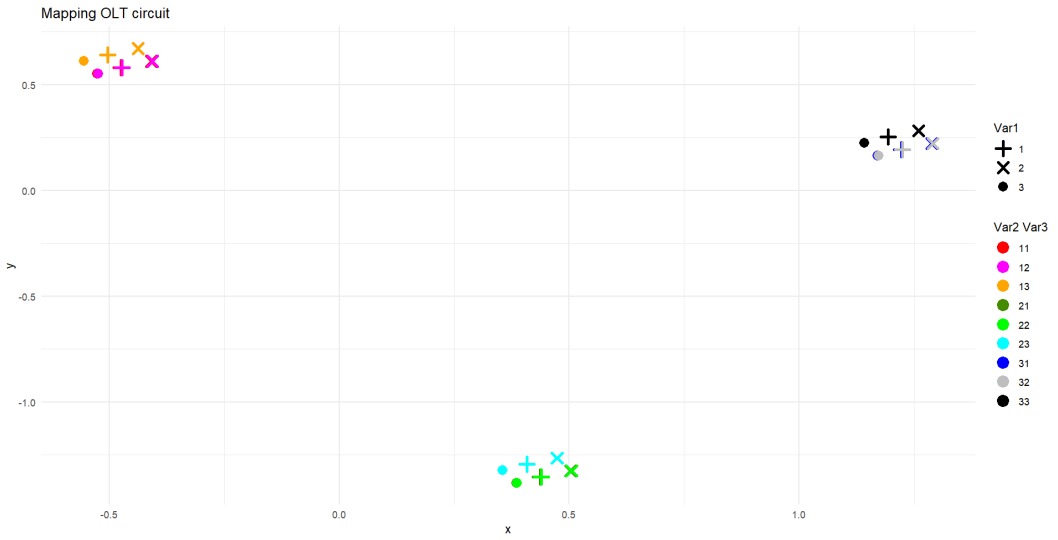

Figure 5.2.3: Latent space, OLT circuit function, 100 samples, MSE = 19×10^{-4}

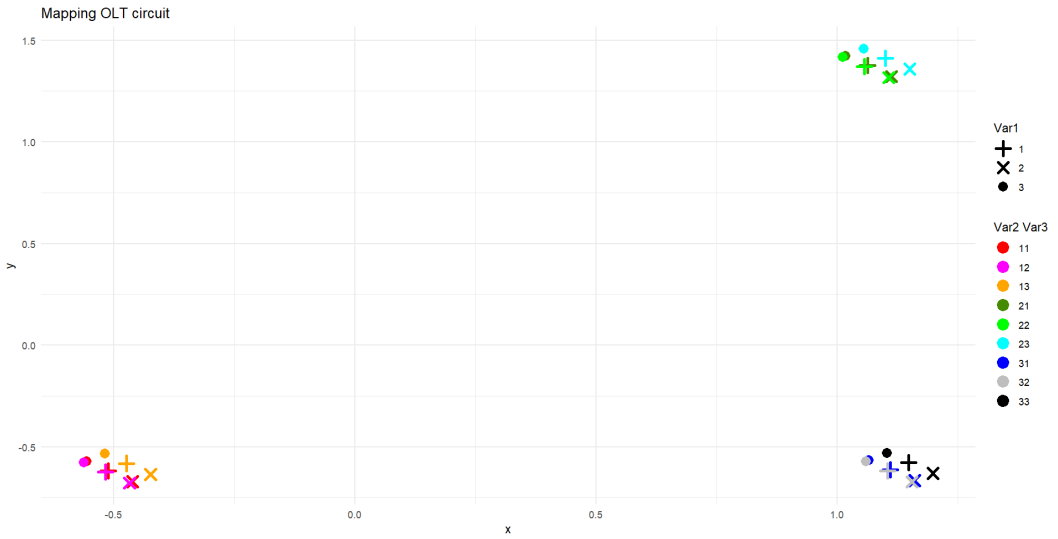

Figure 5.2.4: Latent space, OLT circuit function, 100 samples, MSE = 18×10^{-4}

5.3 Implemented Solutions

We implemented several solutions to address our optimization issues. Here is a summary of what was done:

Unsuccessful Methods:

- To address the problem of rotation of latent spaces, we attempted to impose conditions on the projections, as done in the article [2]. For this, we used the "auglag" function from the R package "nloptr", and we chose L-BFGS to solve the subproblems. The implementation of the method worked in the sense that the constraints were well respected; however, the optimization was very unstable and did not yield better likelihoods than unconstrained methods.

- We also tried using the CMAES optimization algorithm to see if we had better performance, but this was not the case: the optimization took longer for much worse results. However, we did not experiment with many settings of this algorithm.

Successful Methods:

- Standardizing the data, namely the input and output variables of each function, proved to be crucial for obtaining better results. All our best likelihoods were obtained by standardizing the data beforehand.

- Optimization on a logarithmic scale for the parameter ω and the nugget, instead of the classic scale, also improved performance, especially with the Gaussian kernel.

- Multistart (repeating optimizer initializations) allowed for better likelihoods. The authors of the article [2] suggest setting it to 12. We found that this was not sufficient for our implementation. We had to choose a higher multistart when we wanted results similar to those in the article. The larger the sample size increased, the more we noticed that a higher multistart was needed as our optimizer struggled.

- Ordered initialization: We implemented more or less random initializations to better cover the search space of our parameters to optimize during our initialization, with the aim of trying to improve the efficiency of the optimization algorithm. This proved to be crucial. We began improving initialization using a Sobol plan. However, the latter had the disadvantage of not being random. We then decided to use the lhs algorithm, which combines the good distribution of the Sobol plan and randomness.

- Increase in computing power: As a last resort, increasing computing power by parallelizing the code and using clusters with tens of cores, rather than our own machines, allowed us to perform more initializations. This mechanically improved performance.

6 Conclusion

Latent map approach for Gaussian process (LMGP) seems to be an improvement over previous methods for handling categorical variables when using Gaussian process regression. By mapping every level of each categorical variable, this method is more parsimonious than its predecessor LVGP, that map each variable separately. In fact, when comparing Mean Squared Error (MSE), LMGP comes with a better accuracy than LVGP on noisy samples greater than 100 data points.

We found that the Gaussian kernel was generally better than the Matern 5/2 kernel. Additionally, when training an LMGP model, it interpolates the training points. However, achieving exact interpolation becomes more challenging when there are many points, leading to optimization difficulties. Our program also allows optimizing the nugget by treating it as a likelihood parameter. This method has proven to be effective, and we have implemented it in practice. This allowed the LMGP to avoid interpolating the noisy data.

Our implementation of the LMGP method provided satisfying results in low dimensions - 1 to 3 variables, with training samples of 800 observations or less. While large datasets are an inherent issue with Gaussian Processes regression, we encountered some bottlenecks within our program, especially while optimizing the pseudo-likelihood by adjusting the mapping matrix, weights associated with quantitative variables as well as the nugget parameter.

With larger training sets, the optimization grew in complexity, due to what appears to be a recrudescence of local optimum. Indeed, achieving an optimum requires performing multiple initializations during likelihood calculation to avoid falling into a local maximum. However, this becomes more challenging when dealing with a large amount of data and numerous parameters simultaneously. This is due to the program's complexity, which is $O(n^3)$, and the number of required initializations that increases with the dimension of the parameter space.

Even with multiple initialization strategies, such as uniform random start, Sobol sequences or LHS sequences, and several optimizers - BFGS, L-BFGS, Bobyca, CMAES, we encountered issues with optimization that lead to poor overall performance of our implementation. However, with a lot more of initialization - approximately a 1000 instead of the 12 suggested in the paper, we could achieve similar and even better accuracy in some cases, at the cost of higher training time.

Potential area of improvement could involve reinstating invariance constraints and attempting to optimize under those to limit the proportion of local maxima. Another option would be to increase the number of initializations during optimization, going from 12 to a few hundred or even 1000. Although this would significantly increase computation costs, optimization algorithms can be parallelized, which, combined with the initialization strategies mentioned earlier, could allow for better initialization of the optimization algorithms and enable better model convergence overall. The fact that our optimizers used a finite difference method could also explain issues in the convergence, working with exact gradient formula could improve the model.

References

[1] G. Rätsch O. Bousquet U. Von Luxburg. "Advanced lectures on Machine Learning". In: *Springer* (2003), pp. 63–71.

[2] N. Oune and R. Bostanabad. "Latent map Gaussian processes for mixed variable metamodeling". In: *Comput. Methods Appl. Mech. Engrg.* 387 (2021).

[3] C. E. Rasmussen and C. K. I. Williams. "Gaussian Processes for Machine Learning". In: *MIT Press* (2006), p. 16.

[4] Y. Zhang et al. "A latent variable approach to gaussian process modeling with qualitative and quantitative factors". In: *Technometrics* 62.3 (2019), pp. 291–302.

Annex

Documentation of our LMGP program

We have implemented an R package for fitting an LMGP model to data and making predictions based on a fitted model. The computationally intensive parts were programmed in $C++$ using the Rcpp library. This includes matrix inversions, determinant calculations, and for loops that are computationally expensive in R. The R 'optim' function, with the default BFGS method, was used to find the maximum likelihood. For matrix inversions and determinant calculations, we utilized Cholesky decomposition, since the matrices involved were symmetric positive definite - we assumed that no observations were present multiple time in the matrix. We had to use these tricks because Gaussian process fitting has an algorithmic complexity of $\mathrm{O}(n^3)$, which necessitates designing an efficient program.

Our entire program is based on the theory presented above, derived from the article [2]. The kernels that users can choose are the Gaussian kernel and the Matérn $\frac{5}{2}$ kernel. To facilitate the inversion of the matrix R and avoid overfitting, our implementation also allows the addition of a small number to the diagonal of R.

The program has two main functions that the user should use: LMGP.fit and LMGP.predict. The rest of the functions are called by these two but are not intended to be used directly by the user. Here is the documentation for the two main functions:

Function LMGP.fit

The `LMGP.fit` function is used to fit an LMGP model to hybrid data (both character and numeric). This function takes the data and several optional parameters as input and returns a fitted model.

Function Parameters

- `data_frame_type (data.frame, size` $n \times p$`)`: The raw dataframe containing explanatory variables, with columns of type "numeric" or "character".
- `Y_vec (numeric, size` n`)`: The response vector.
- `N_try (integer)`: The number of initializations to perform during optimization.
- `kernel_str (character)`: The kernel to use ("exp" for exponential or "mat" for Matern).
- `F_matrix (matrix, size` $n \times h$`)`: The base matrix F described in the paper. Default is the $n \times 1$ matrix $(1)_{i \in [\![1,n]\!]}$.

- `ncol_A (integer)`: The number of columns in matrix A describing the dimension of the projection from the qualitative matrix to the latent space.
- `reg (numeric, size 1)`: The constant to add to the diagonal of matrix R (smoothing parameter). Default is NA. This means that the nugget is optimized by maximum likelihood.
- `silent (logical)`: Flag to enable/disable messages during execution.
- `scale (logical)`: Flag to enable/disable data scaling.
- `amplitude_A (numeric, size 2)`: Initialization amplitude for elements of matrix A.
- `amplitude_w (numeric, size 2)`: Initialization amplitude for elements of vector ω.
- `amplitude_reg (numeric, size 2)`: Initialization amplitude for the nugget.
- `optimiseur (character)`: The optimization method ("L-BFGS-B","BFGS","bobyka","cmaes").
- `max_it_optimization (integer)`: Maximum number of iterations in one optimization try.
- `relative_tol_optimization (integer)`: Relative tolerance for the cost function optimization.
- `pow_w_10 (logical)`: Should optimization be performed on a logarithmic scale?
- `type_init (character)`: The initialization method ("lhs","sobol","unif").

Function Result

- `model (list)`: The fitted model, encoded in a list, ready for direct use in the LMGP.predict function.

Potential Errors

- `No Convergence`: If none of the N_try initializations converges, an error is returned.
- `Ineffective Model`: If the model predicts less accurately than the simple mean of the Y_vec vector in terms of MSE, an error is returned.

Function LMGP.predict

The `LMGP.predict` function is used to make predictions from new data using a pretrained model. It predicts the mean and variance of the response conditionally on the predictors.

Function Parameters

- `fitted_LMGP (list)`: The model returned by the `LMGP.fit` function.
- `new_data_frame_type (data.frame, size` $k \times p$`)`: The new raw dataframe containing explanatory variables, with columns of type "numeric" or "character".
- `new_F_matrix (matrix, size` $k \times h$`)`: The new base matrix F described in the paper. Note that the functional basis used to construct F should be the same as the one used in the `LMGP.fit` function.

Function Result

- `predictions (list)`: The first element of the list contains the predicted mean, and the second element contains the predicted variances.

OLT Circuit formula

The OLT circuit formula is given by:

$$y = \frac{(V_{b1} + 0.74)\beta(R_{c2} + 9) + 11.35R_f}{(\beta(R_{c2} + 9) + R_f)R_{c1}} + \frac{0.74R_f\beta(R_{c2} + 9)}{(\beta(R_{c2} + 9) + R_f)R_{c1}} \tag{1}$$

Where $V_{b1} = \frac{12R_{b2}}{R_{b1}+R_{b2}}$

Borehole formula

The Borehole formula is given by:

$$y = \frac{2\pi T_u(H_u - H_l)}{\ln\left(\frac{r}{r_\omega}\right)\left(1 + \frac{2LT_u}{\ln(\frac{r}{r_\omega})r_\omega^2 K_\omega} + \frac{T_u}{T_l}\right)} \tag{2}$$

Custom function formula

We also tested the custom function, which formula is given by:

$$y = 4(x_1 - 2 + 8x_2 - 8x_2^2)^2 + (3 - 4x_2)^2 + 16\sqrt{x_3 + 1}(2x_3 - 1)^2 + \sum_{i=4}^{n} i \times \ln\left(1 + \sum_{j=1}^{i} x_j\right) \tag{3}$$

Benchmark including LMGP and RF

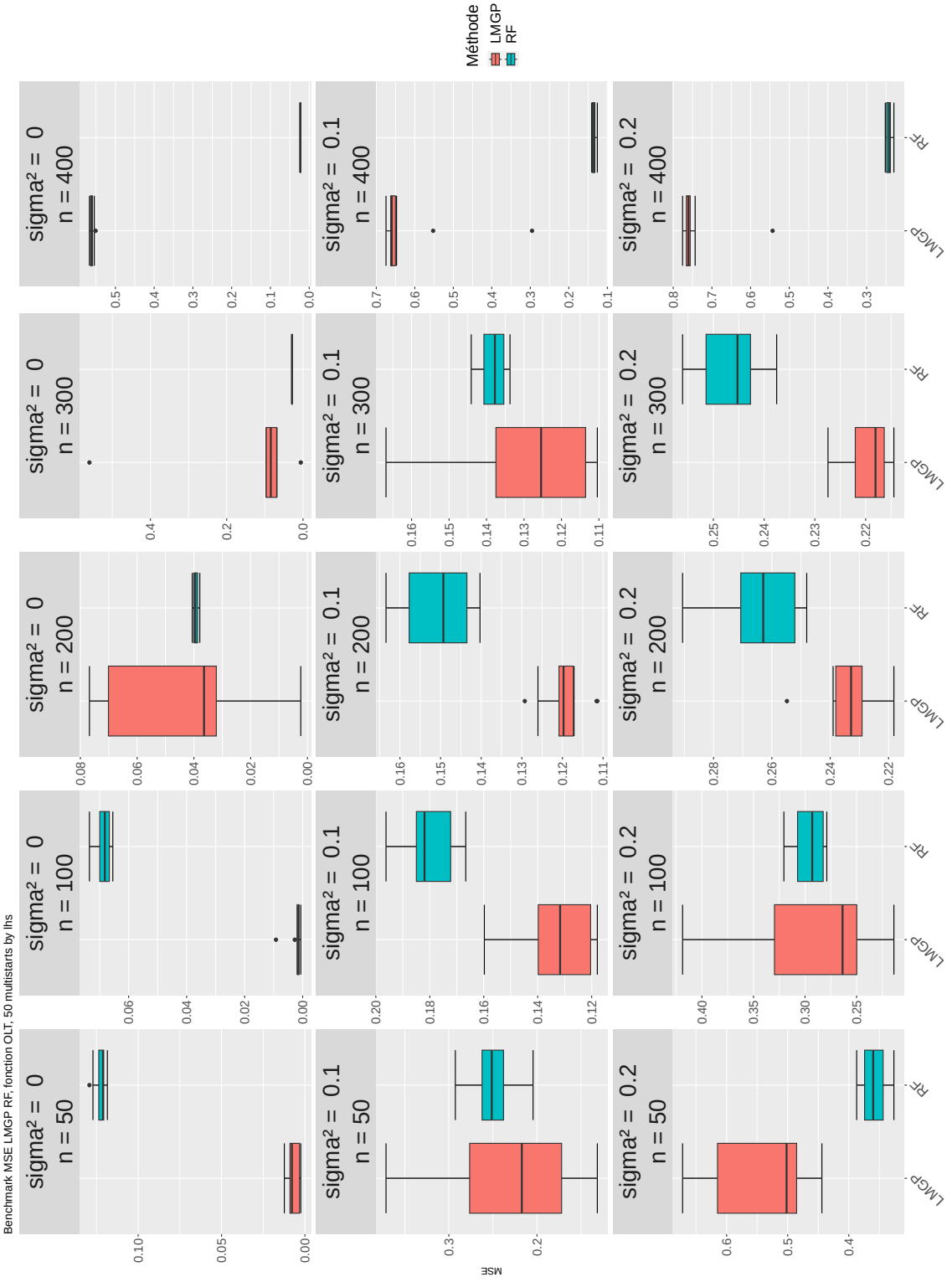

Figure 6.0.1: Results of LMGP and RF benchmark on the OLT circuit function, simulated data, 50 multistarts with LHS initialization

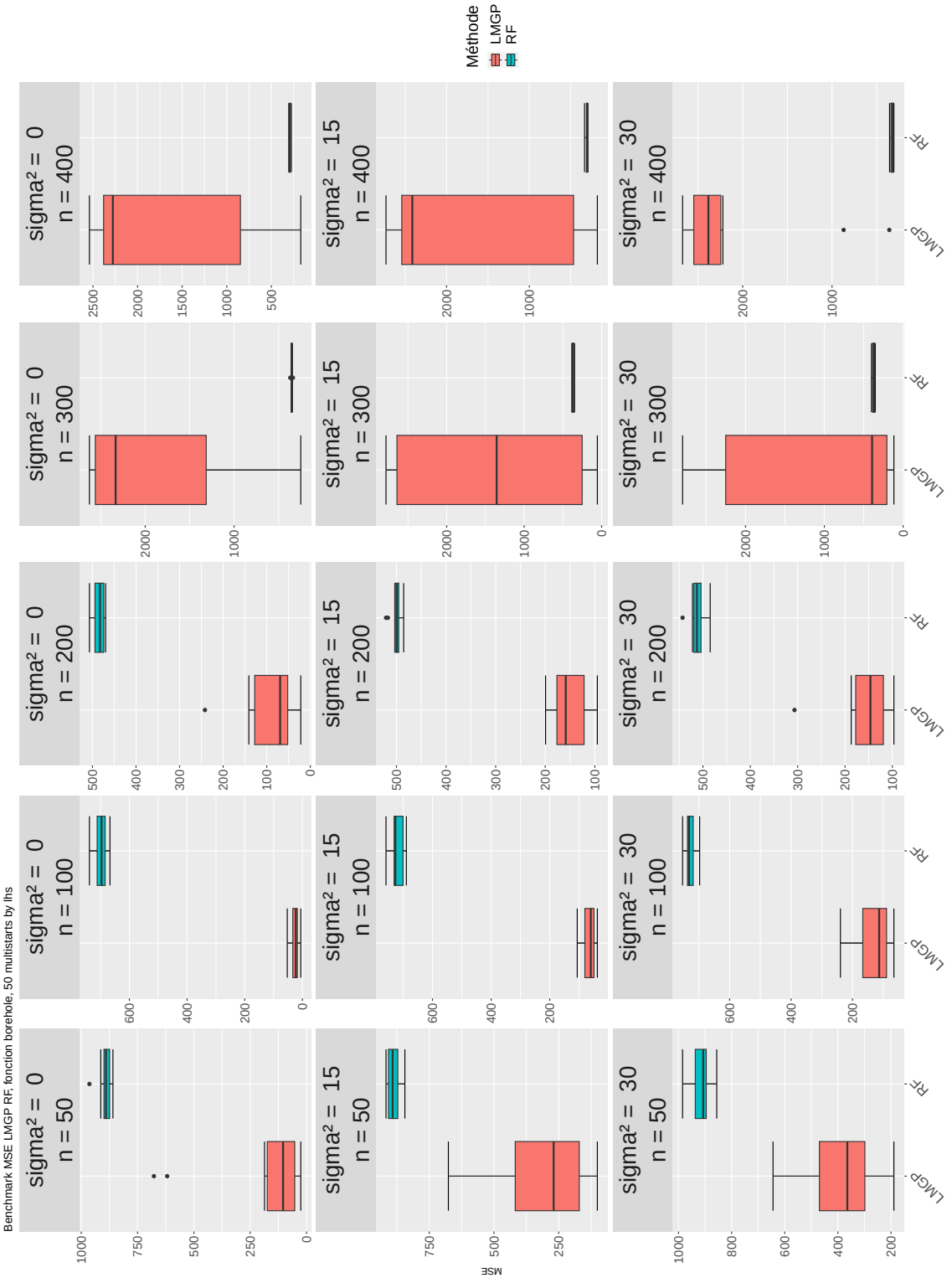

Figure 6.0.2: Results of LMGP and RF benchmark on the borehole function, simulated data, 50 multistarts with LHS initialization

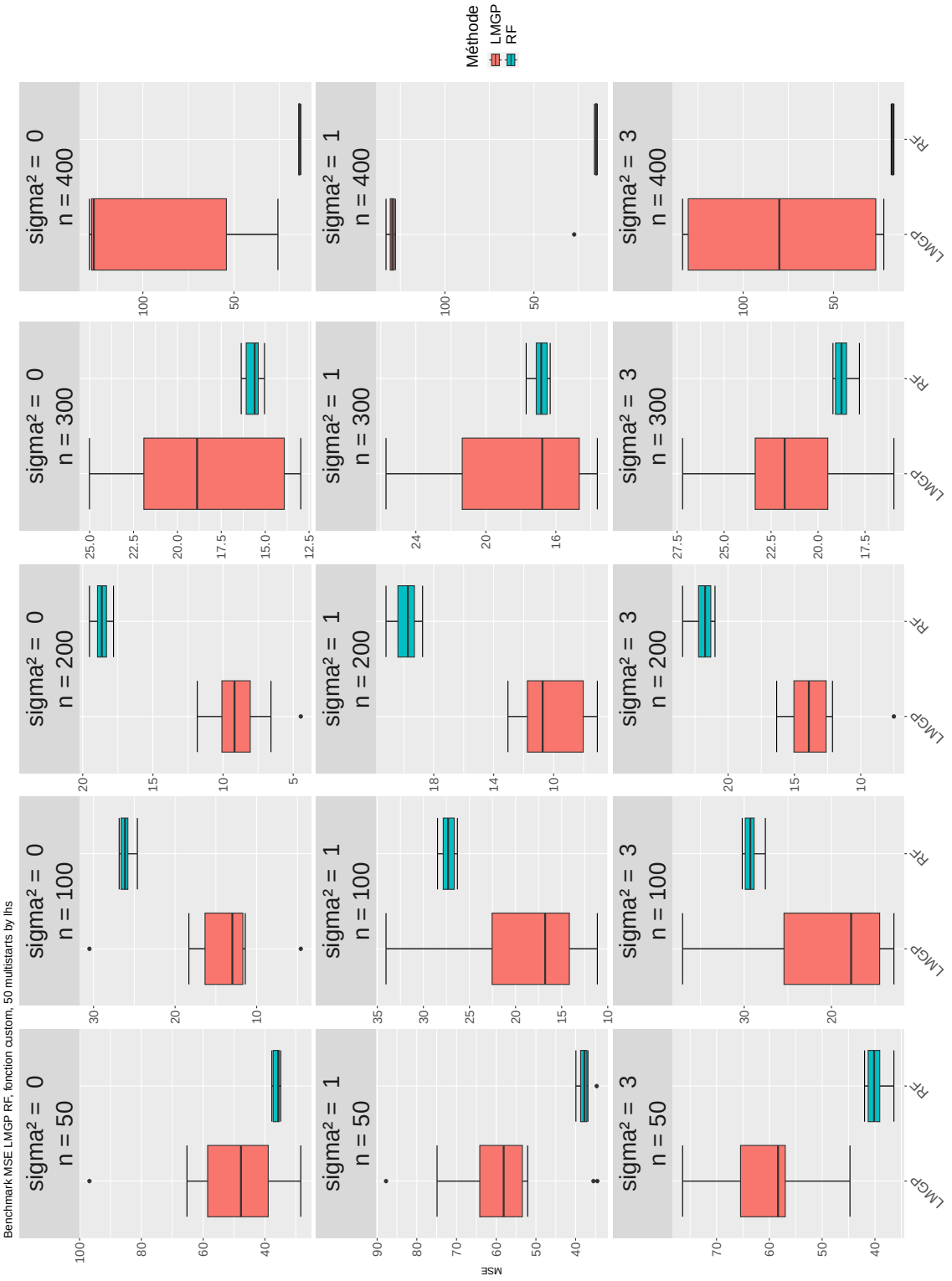

Figure 6.0.3: Results of LMGP and RF benchmark on the custom function, simulated data, 50 multistarts with LHS initialization

Fitting times

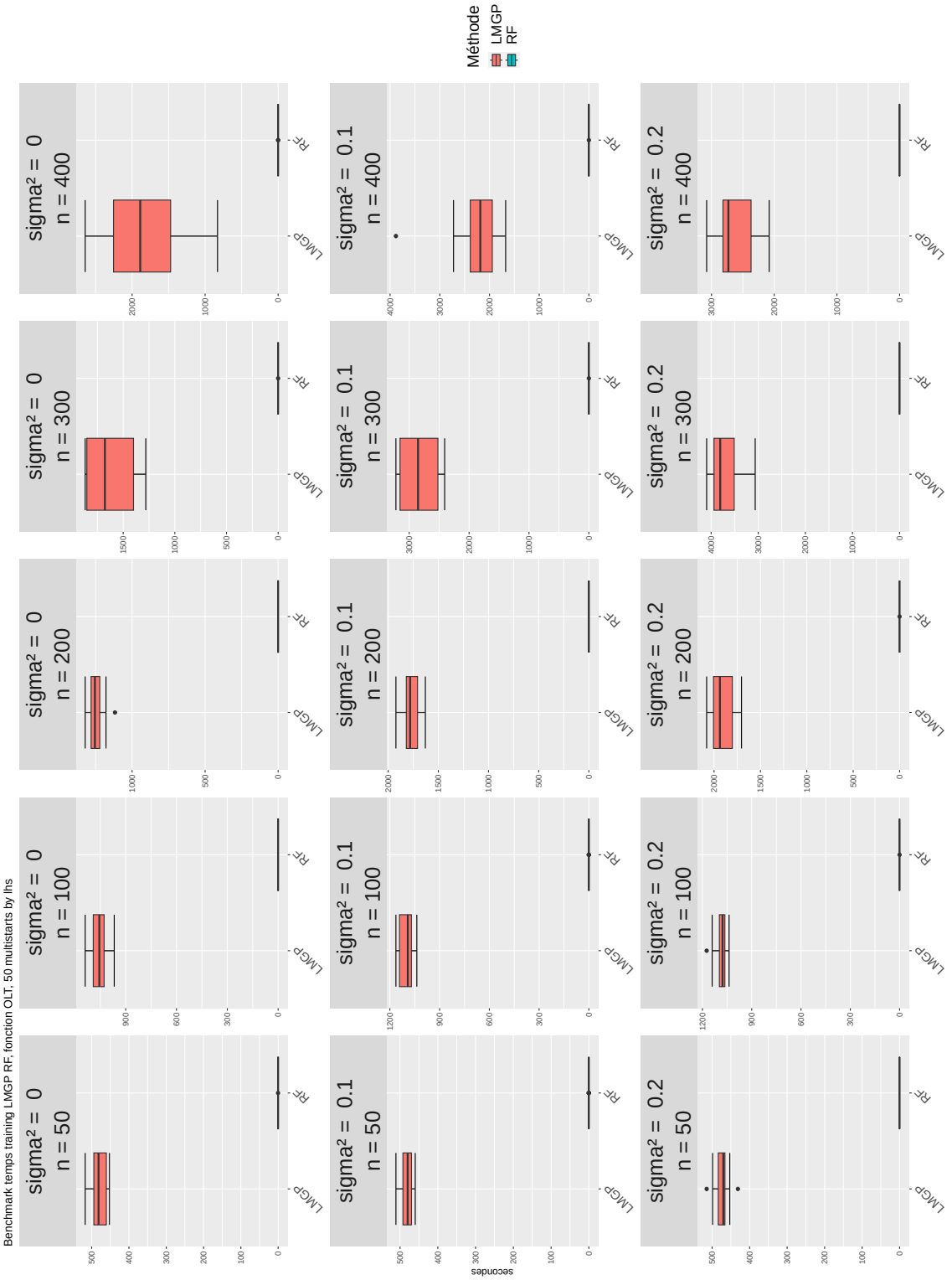

Figure 6.0.4: Fitting times in seconds of LMGP and RF benchmark on the OLT circuit function, simulated data, 50 multistarts with LHS initialization

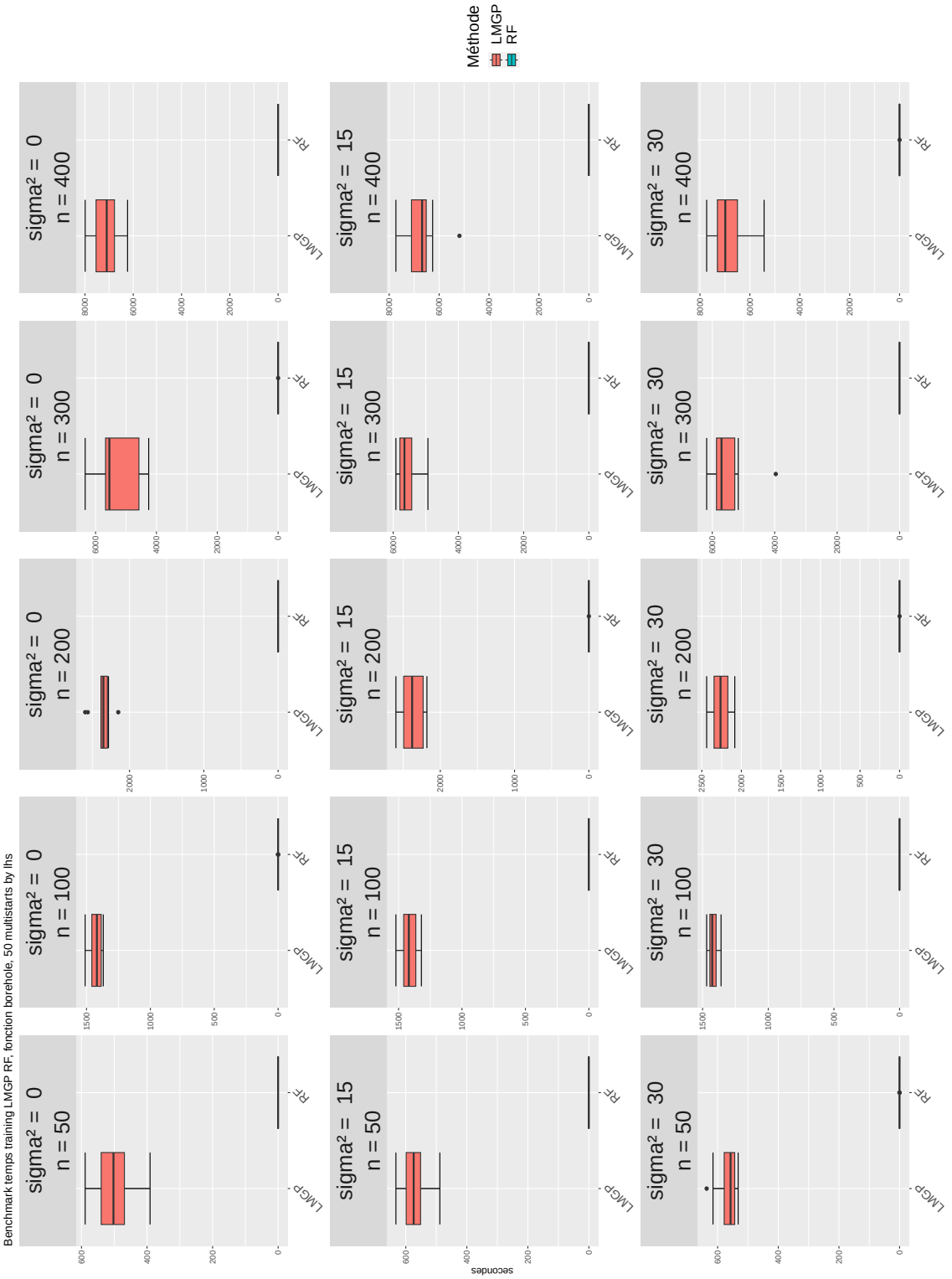

Figure 6.0.5: Fitting times in seconds of LMGP and RF benchmark on the borehole function, simulated data, 50 multistarts with LHS initialization

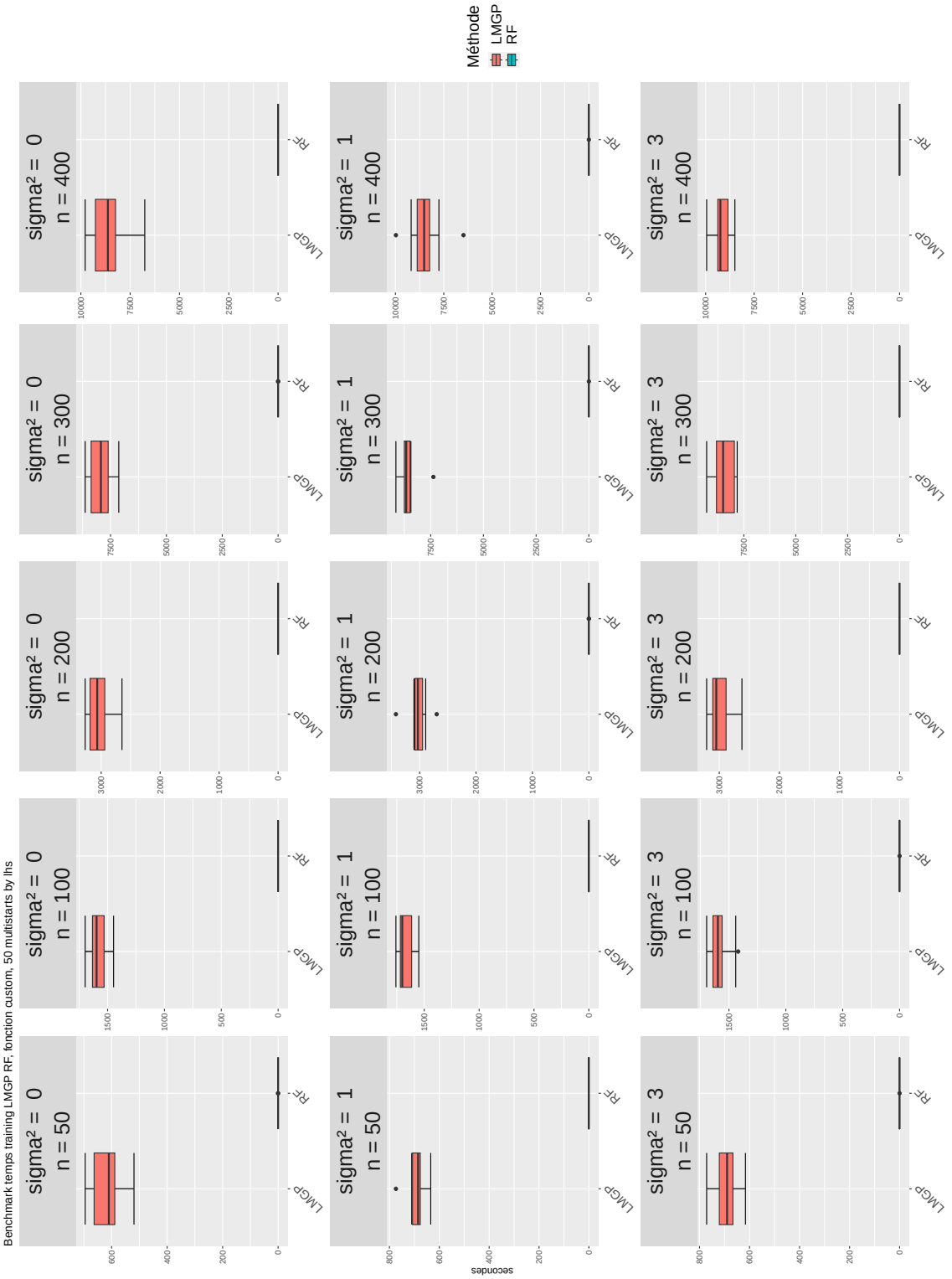

Figure 6.0.6: Fitting times in seconds of LMGP and RF benchmark on the custom function, simulated data, 50 multistarts with LHS initialization