École Polytechnique Fédérale de Lausanne

Gaussian Process Regression

MATH-414: Stochastic Simulation

Francesco Pettenon
Francesca Venturi

11th January 2023

1 Introduction

1.1 Context

This project finds its greatest motivation in the field of geoscience, where approximating unknown properties of the subsurface by means of Gaussian processes is a standard procedure. In particular, we know that scientists have at their disposal limited data collected through boreholes. The goal of the project is therefore to derive information about the entire permeability field of the area being studied by means of Gaussian Process Regression, that is, to make use of conditional Gaussian processes in order to reliably predict the unknown permeability field.

1.2 Motivation

Gaussian process regression is a nonparametric method for regression that includes uncertainty quantification in its predictions and performs well on small datasets. It is based on Bayesian statistics, which assumes that the parameters describing the unknown field follow a known distribution (called the prior distribution) and that the observations also follow a certain distribution. Using Bayes' Rule, the prior distribution is updated based on the observations to obtain the posterior distribution, which incorporates information from both the prior and the observations. The predictive distribution is then obtained by integrating the posterior distribution and it is Gaussian as well. This method does not provide pointwise predictions, but rather a distribution from which single values can be derived using the mean and covariance matrix [1].

2 Theoretical Background

This project focuses on the generalization of the above procedure to Gaussian processes and Gaussian random fields, known as Gaussian Process (GP) regression. The theoretical background is analyzed in detail in [1] and [2], which we have referred to throughout the entire project. For convenience, we list below the essential formulas and their notation. Let $\mathbf{f}(Z) \sim \mathcal{N}(\mathbf{m}(Z), \mathbf{K}(Z))$ be a Gaussian random field, and denote with Z any finite set of positions and Y any new set of locations, whose value $\mathbf{f}(Y)$ is to predict. Furthermore assume that the we deal with noisy observations $\tilde{\mathbf{f}}(Z) = \mathbf{f}(Z) + \epsilon$ where $\epsilon \sim \mathcal{N}\left(\mathbf{0}, s^2\mathbf{I}_{N_Z}\right)$, then we know that the covariance of $\tilde{\mathbf{f}}(Z)$ is $\mathbf{K}(Z) + s^2\mathbf{I}$. Therefore, we have that $\mathbf{f}(Y)|\tilde{\mathbf{f}}(Z) \sim \mathcal{N}(\tilde{\mathbf{m}}(Y \mid Z), \mathbf{K}(Y|Z))$, whose mean and covariance matrix are:

$$\tilde{\mathbf{m}}(Y \mid Z) := \mathbb{E}[\mathbf{f}(Y) \mid \tilde{\mathbf{f}}(Z)] = \mathbf{m}(Y) + \mathbf{K}(Y, Z)\left(\mathbf{K}(Z) + s^2\mathbf{I}\right)^{-1}(\tilde{\mathbf{f}}(Z) - \mathbf{m}(Z)) \tag{1}$$

$$\mathbf{K}(Y \mid Z) = \mathbf{K}(Y) - \mathbf{K}(Y, Z)\left(\mathbf{K}(Z) + s^2\mathbf{I}\right)^{-1}\mathbf{K}(Z, Y). \tag{2}$$

In our project we always assume the prior means $\mathbf{m}(Y) = 0$ and $\mathbf{m}(Z) = 0$.

2.1 Covariance kernels

In the following study, we will make use of two covariance kernels, that is the tool for modeling covariance and the relationship between two variables. Both kernels we present are typically used to model stationary random processes. Recalling that $\|\cdot\|_2$ denotes the Euclidean norm of a vector, we introduce both kernels:

1. The Exponential (EXP) kernel given by

$$K_{exp}\left(\mathbf{x}, \mathbf{x}'\right) = \sigma^2 \exp\left\{-\frac{\|\mathbf{x} - \mathbf{x}'\|_2}{\ell}\right\}, \tag{3}$$

2. The Squared Exponential (SE) kernel given by

$$K_{se}\left(\mathbf{x}, \mathbf{x}'\right) = \sigma^2 \exp\left\{-\frac{1}{2}\sum_{i=1}^{k}\frac{(x_i - x_i')^2}{\ell_i^2}\right\} \tag{4}$$

In both of these expressions, we denote by σ^2 the variance and by $\ell = (\ell_1, ..., \ell_k)$ the correlation length or lengthscale. Although their formulation is similar, some considerations must be done.

One key difference between the two kernels is that the SE kernel gives more weight to points that are closer together by means of the square exponent, resulting in a smoother function. The EXP kernel, on the other hand, presents a linear exponent with respect to the distance between two points and this can lead to a rougher function. To be clearer, we plot two covariance matrices that are generated from the very same dataset, while they only differ with respect to the kernel employed: we can observe that the SE kernel has a larger correlation band around the diagonal, meaning that it actually gives more weight than the EXP to the points that are closer.

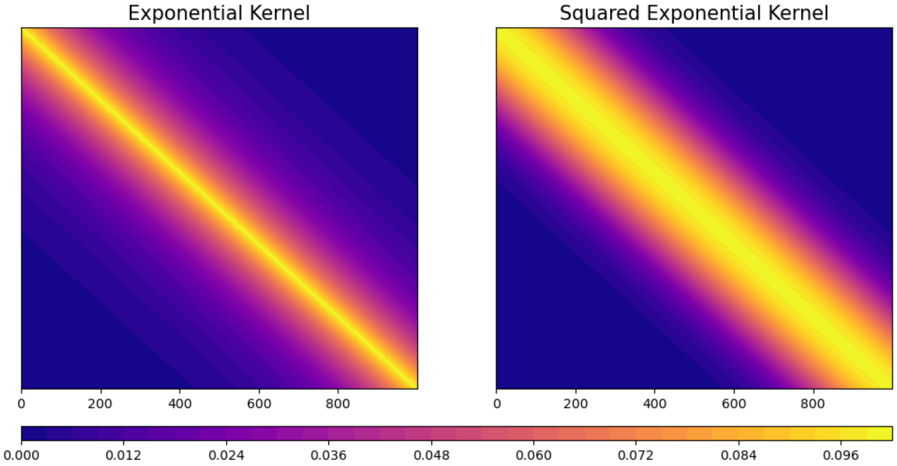

FIGURE 1
Visual difference between the EXP and the SE kernels 1D with fixed parameters σ^2 and ℓ. These two matrices are taken as an example from Section (3) for a qualitative trend.

Another important difference arises in the case where the problem being studied is multimensional (see Section 4). On the one hand, the exponential kernel, being by construction isotropic, only has one value for ℓ, whereas, instead, in the case of squared exponential kernel the correlation length becomes vector, thus admitting anisotropy. The isotropy of a kernel refers to whether the kernel is sensitive to the orientation of the data or not: as a consequence, the former is equally sensitive to the data in all directions, while the latter can capture directional patterns.

HYPERPARAMETERS: ℓ AND σ^2

The Correlation Length ℓ: in both the EXP kernel and the SE kernel, the correlation length determines the degree of smoothness in the function being modeled. However, their interpretation is slightly different. In the exponential kernel, ℓ controls the rate at which the function being modeled drops off as the distance between two points increases. A larger correlation length means that the function will drop off more slowly, resulting in a smoother shape. A larger correlation length means that the function can vary over a larger length scale, resulting again in a smoother function.
The Vertical Scale σ^2: the vertical spread of a function is determined by its variance. The vertical scale σ^2 dictates the degree to which the predicted function deviates from the average prediction at any given point. A high variance implies greater flexibility, allowing the function to fluctuate more, while a low variance means that the function is restricted and will have less variations.

3 Recovering a simple function

The preliminary stage of the study concerns recovering a scalar function. Indeed, suppose that the data we want to predict are generated by a simple function such as $f(x) = sin(x)$ and further assume that the N_Z observations we have available are noise-free, i.e., that $y = sin(x)$. The goal of this first task is then to predict the entire Random Process, i.e., to generate N_Y gaussian random variables, whose mean correctly approximates the sinusoidal function above.

3.1 Gaussian Process Generation

At the operational level, we proceed as follows.

- The domain of interest is the period of the function $sin(x)$ i.e., the interval $[0, 2\pi]$, from which we randomly extract N_Z values Z_i for $i = 1, ..., N_Z$, at which we evaluate the function $sin(Z_i)$, thus obtaining the Gaussian vector $\mathbf{f}(Z) \sim \mathcal{N}(\mathbf{m}(Z), \mathbf{K}(Z))$, where $Z = \{Z_i\}_{i=1}^{N_Z}$, $\mathbf{m}(Z) = sin(Z)$ and $\mathbf{K}(Z) = \mathbf{K}(Z, Z)$. Please note that, at this level, there is no difference between the two kernels and so, for now, we stick to a more general notation where the function $K(\mathbf{x}, \mathbf{x}')$ represents a generic kernel. We will look at the difference between K_{exp} and K_{se} when we analyze the results obtained (Section 3.2). Aiming to replicate the performance of a continuous function, we decide to divide the same interval into a sufficiently fine partition to hit the target, thus selecting a 1D uniform grid of $N_Y = 1000$ values, which we identify with Y_i for $i = 1, ..., N_Y$.
- On the basis of what has already been presented in the section on theoretical analysis of Gaussian processes (Section 2), our intention is to study the conditional distribution $\mathbf{f}(Y)|\mathbf{f}(Z)$, of which we know the mean (1) and covariance (2). From these formulas, it is evident that, in order to generate the conditional Gaussian process, we need to compute the covariance matrices $\mathbf{K}(Z), \mathbf{K}(Y), \mathbf{K}(Y, Z)$. In this regard, therefore, we evaluate the function $K(\mathbf{x}, \mathbf{x}')$ for the vectors Z and Y.
- Once the kernels have been evaluated, all that remains is to calculate the mean and variance of the conditional process. We note that although the observations are noise-free, $s^2 > 0$ is still considered. Indeed, we know that the computation of an inverse matrix can be numerically instable when the eigenvalues of the original matrix are close to zero. The term we add, therefore, allows each eigenvalue of that matrix to be increased by s^2, thus stabilizing the numerical procedure. Once the conditional mean and covariance are obtained, we observe that the covariance is nearly singular, so we proceed by generating the 3 separate Gaussian random processes using SVD, as described in [[2], Algorithm 3.1].

The whole procedure that we have just described is formally summarized in Algorithm 1, where the further step of optimizing the hyperparameters of the problem is performed, too.

3.2 Results

In this section we are concerned with analyzing the newly generated Gaussian processes and how they vary according to the amount of observations, i.e. N_z, and the choice of kernel.

Based on the plots generated in Figure (2-3), there are notable considerations:

- First, as is to be expected, the confidence interval decreases considerably as N_Z increases: by adding information to the conditioning process (i.e., by increasing the vector of observations on which the model is trained) more meaningful information of the conditional distribution is obtained. This leads to a considerable decrease of the predictive uncertainty.

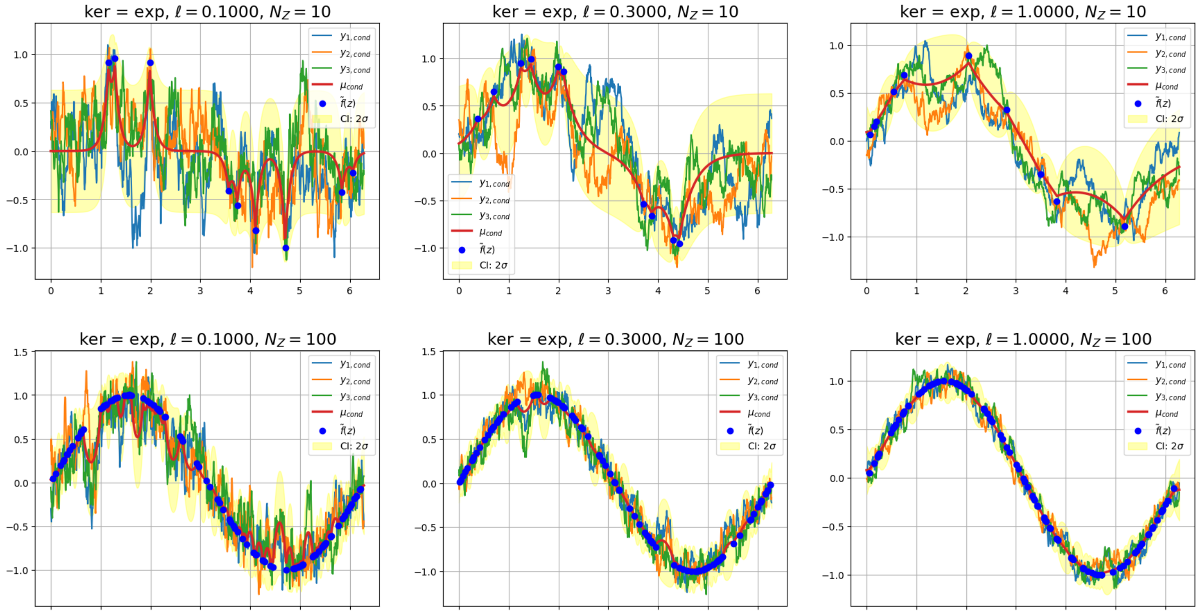

FIGURE 2
1D Gaussian Process: Exponential Kernel

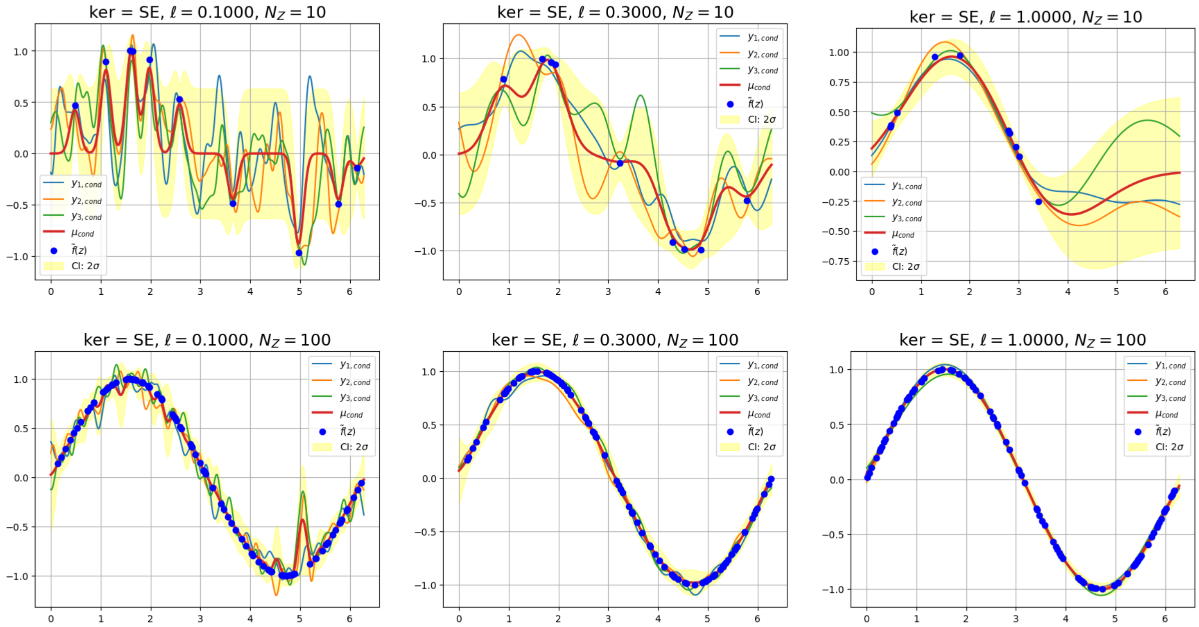

FIGURE 3
1D Gaussian Process: Squared Exponential Kernel

- The most interesting observations, however, are about correlation length and the impact it has on posterior distribution. As noted in (2.1) the parameter ℓ characterizes the smoothness of the function being modeled. Indeed, from the plots, it is evident that as correlation length increases, the smoothness of the conditional mean (red line) increases. For example, for ℓ=0.1, for the same distance between the variables, there is a poorer covariance: this implies that the posterior mean of the Gaussian processes does not deviate from the prior mean (zero in this case), except for the points at the Z_i observations.
- As anticipated, the SE kernel assumes more correlation between points close to each other, so it

physiologically performs slightly better in our case.

- Finally, we observe that the confidence interval width does not cancel at Z_i observations precisely because of the addition of the s^2 stabilizing term mentioned above.

3.3 Optimize hyperparameters

In general, the specific parameters $\left\{\sigma^2, \ell, s^2\right\}$ that appear in the expression of the covariance kernels for GP regression can greatly impact the performance of the model. When no prior knowledge is available about their values, one approach to selecting them is to maximize the marginal likelihood

$$p_{\tilde{\mathbf{f}}(Z)}(v) = \int p_{\tilde{\mathbf{f}}(Z)|\mathbf{f}(Z)}(v \mid \mathbf{f}) p_{\mathbf{f}(Z)}(\mathbf{f}) \mathrm{d}\mathbf{f}, \text{ where } \begin{cases} p_{\tilde{\mathbf{f}}(Z)|\mathbf{f}(Z)}(\cdot \mid \mathbf{f}) = \mathcal{N}\left(\mathbf{f}, s^2\mathbf{I}\right) \\ p_{\mathbf{f}(Z)}(\cdot) = \mathcal{N}(\mathbf{0}, \mathbf{K}(Z)) \end{cases} \tag{5}$$

From this, one can compute the integral and obtain the marginal log-likelihood

$$L = \log p_{\tilde{\mathbf{f}}(Z)}(v) = -\frac{1}{2} v^T \left(\mathbf{K}(Z) + s^2\mathbf{I}\right)^{-1} v - \frac{1}{2} \log\left|\det\left(\mathbf{K}(Z) + s^2\mathbf{I}\right)\right| - \frac{N_Z}{2} \log 2\pi \tag{6}$$

As a result of the analytical computation of the marginal log-likelihood gradients (C), we obtain:

$$\frac{\partial L}{\partial s^2} = \frac{1}{2} v^T \left[\tilde{\mathbf{K}}^{-1} \tilde{\mathbf{K}}^{-1}\right] v - \frac{1}{2} \operatorname{Tr}\left(\tilde{\mathbf{K}}^{-1}\right) \tag{7}$$

$$\frac{\partial L}{\partial \sigma^2} = \frac{1}{2} v^T \left[\tilde{\mathbf{K}}^{-1} \mathbf{K}_{\sigma^2} \tilde{\mathbf{K}}^{-1}\right] v - \frac{1}{2} \operatorname{Tr}\left(\tilde{\mathbf{K}}^{-1} \mathbf{K}_{\sigma^2}\right) \tag{8}$$

$$\frac{\partial L}{\partial \ell_i} = \frac{1}{2} v^T \left[\tilde{\mathbf{K}}^{-1} \mathbf{K}_{\ell_i} \tilde{\mathbf{K}}^{-1}\right] v - \frac{1}{2} \operatorname{Tr}\left(\tilde{\mathbf{K}}^{-1} \mathbf{K}_{\ell_i}\right) \tag{9}$$

Finally, we proceed by numerical optimization of the parameters by means of the gradient-based optimization algorithm `L-BFGS-B`, which yields the following values:

$N_Z = 10$	ℓ	σ^2	s^2
EXP	3.406	0.411	0.000
SE	2.041	0.444	0.000

$N_Z = 100$	ℓ	σ^2	s^2
EXP	12.758	0.182	0.000
SE	1.146	1.089	0.001

Table 1
Optimized hyperparameters

The most important feature is the optimized null value of s^2, from which we infer that the model learns that the observations on which it conditions are noise-free, so it recovers this feature even in the optimized case.

With the optimized values, we then resubmit the previous plots to visually capture the differences and improvements made by this procedure (Figure 4).

It thus becomes apparent how the absence of noise is reflected in the width of the confidence interval at the observations, which is now zero. Moreover, it is evident from the plots how the performance of the regression is extremely better, especially at a smaller dataset size. To conclude, the SE kernel, due to its smooth nature, will be more accurate in learning the smooth function $sin(x)$. On the other hand, the mean learnt by the EXP kernel tends to linearly interpolate between the training points.

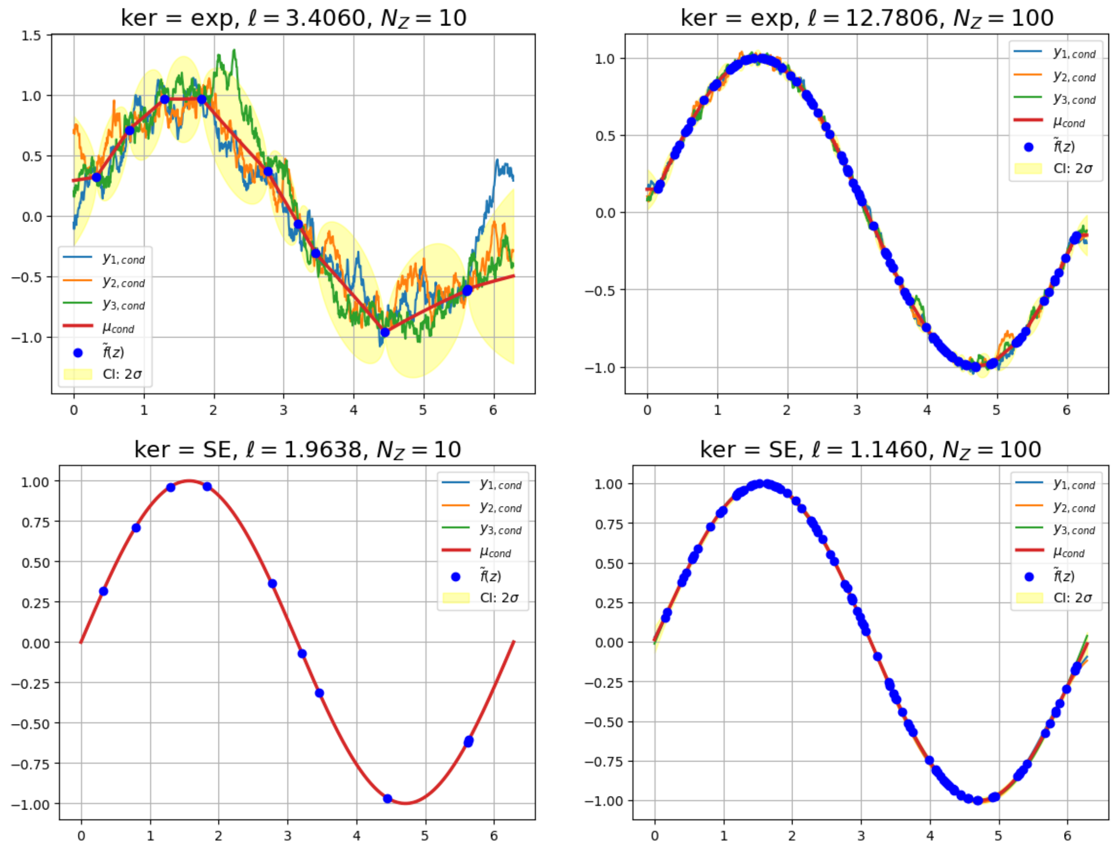

FIGURE 4
Predictions and observations with optimized hyperparameters

The procedure we have just introduced generalizes 1D Gaussian Process Regression and is formalized in the following algorithm:

Algorithm 1 1D Gaussian Process Regression

```
Generate $N_Z$ Points Z uniformly distibuted in $[0, 2\pi]$
Generate 1000 Points Y in a 1D uniform grid of $[0, 2\pi]$
Given $f(x) = sin(x)$, evaluate the Points Z, obtaining $\mathbf{f}(Z)$
Optimize the parameters $\boldsymbol{\theta} = \left(\ell, \sigma^2, s^2\right)$ of the kernel w.r.t. marginal likelihood
Build the covariance matrices $\mathbf{K}(Y)$, $\mathbf{K}(Z)$, $\mathbf{K}(Y, Z)$
Compute the prediction $\mathbf{m}(Y|Z)$ and $\mathbf{K}(Y|Z)$ using (1) and (2)
Draw 3 indipendent realizations of the Gaussian process $\mathbf{f}(Y)|\mathbf{f}(Z)$
```

4 GP Regression on a permeability field

4.1 Dataset and computational considerations

In the second phase of the study, we apply the concepts learned in Section (3) to a geoscience problem in which we aspire to infer unknown properties of the subsurface permeability field based on a limited amount of available data. Specifically, we are provided with 60×110 observations subjected to noise $\epsilon \sim \mathcal{N}\left(\mathbf{0}, s^2\mathbf{I}_{N_Z}\right)$, representing the entire field. The goal of the study is to approximate this field by means of a much smaller number of data, which are used as training set for model generation. In particular, we will focus on two uniform rectilinear grids:

- The first dataset extracts an equispaced grid of 5×5 points, whose coordinates along x are 10, 20, 30, 40, 50 and those along y are 15, 35, 55, 75, 95
- The second dataset, on the other hand, consists of a grid of 11×11 observations, whose coordinates along x are 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55 and along y are 5, 15, 25, 35, 45, 55, 65, 75, 85, 95

The choice to analyze less data than the total amount is evidently dictated by the interest in generating a reliable predictive model that can perform properly once tested on a new sample, decreasing the cost of data collection and the computational complexity of the problem.

Before we begin the actual analysis of the problem, however, it is necessary to manipulate the data we have. In fact, the problem we are going to analyze is in two dimensions: we need to refer to a one-dimensional vector of observations, which allows us to apply the techniques analyzed in the Section (3). All the matrices we will deal with are inspected in *Row-Major Order*: elements of a multi-dimensional array are arranged sequentially row by row, which means filling all the index of first row and then moving on to the next row. What is thus obtained are three vectors: the entire permeability field `true_perm` consisting of 6600 real observations, the first dataset `Z1` of 25 points and the second dataset `Z2` of 121 points.

4.2 Optimize hyperparameters

In order to optimize the hyperparameters, we use the same method as before, i.e., marginal log-likelihood maximization, obtaining the following results. Notice that, in this case, the SE kernel has two correlation length values, thus it is able to pick up any directional trend in the permeability field.

Z1	σ^2	s^2	ℓ_1	ℓ_2
EXP	12.395	1e-10	64.590	-
SE	9.047	0.589	16.330	21.891

Z2	σ^2	s^2	ℓ_1	ℓ_2
EXP	11.428	0.035	40.395	-
SE	7.005	1.120	11.720	18.266

Table 2
Optimal hyperparameters $\theta = \{\sigma^2, s^2, \ell, \}$

4.3 Predict the Gaussian Process

Once the parameters have been optimized, it is possible to obtain the matrices $\mathbf{K}(Y)$, $\mathbf{K}(Z)$, $\mathbf{K}(Y, Z)$ for both kernels and both domains. Finally, by exploiting the formulas for the distribution of conditional Gaussian random variables, the mean $\tilde{\mathbf{m}}(Y \mid Z)$ and covariance matrix $\mathbf{K}(Y|Z)$ can be computed. So, for each kernel and for each dataset, the algorithm is:

Algorithm 2 2D Gaussian Process Regression

```
Create 2D-meshgrid with Points $Z$
Extract the Points Z in true_perm, obtaining $\mathbf{f}(Z)$
Create the link between the physical points (dim = $(110 \times 60)$) and the Gaussian Vector $Y$ (dim = 6600):
    $Y[n] \iff \texttt{Points}[n/60][n\%60] \quad \text{and} \quad \texttt{Points}[i][j] \iff Y[i \cdot 60 + j]$
Optimize the parameters $\boldsymbol{\theta} = (\boldsymbol{\ell}, \sigma^2, s^2)$ of the kernel w.r.t. marginal likelihood
Build the covariance matrices $\mathbf{K}(Y)$, $\mathbf{K}(Z)$, $\mathbf{K}(Y, Z)$
Compute the prediction $\tilde{\mathbf{m}}(Y \mid Z)$ and $\mathbf{K}(Y|Z)$ using (1) and (2)
return $\tilde{\mathbf{m}}(Y \mid Z)$, $\mathbf{K}(Y|Z)$
```

In Figure (5 - 6) we plot, for each kernel, the mean and variance of the approximate permeability field at both datasets of observations.

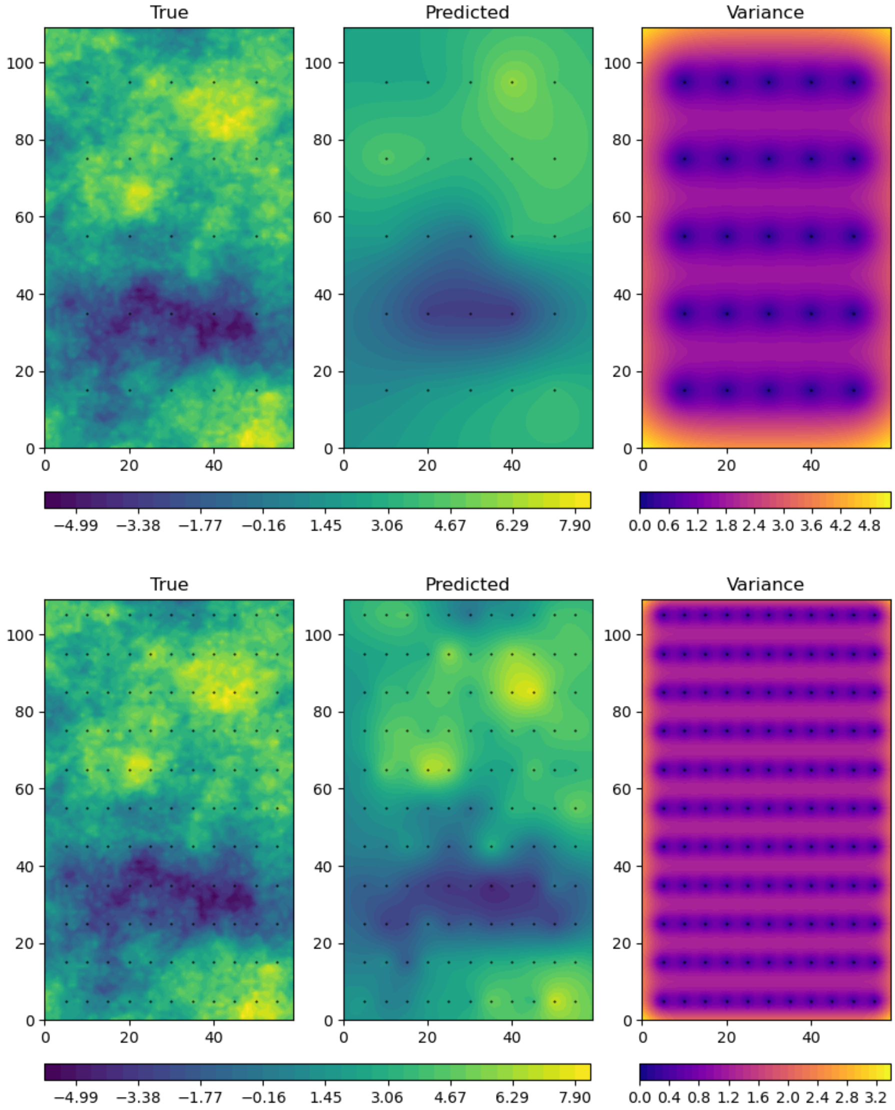

FIGURE 5

2D Gaussian Process: Exponential Kernel

Although plots in two dimensions are inherently less intuitive, some noteworthy behaviors can be detected.

- First, it is observed that by increasing the dimension of the training set, the variance of the observations falls, which means that the confidence interval of each prediction narrows. Intuitively, as the number of available data increases, the performance of the model improves, decreasing its uncertainty.
- In the plots of the variances we observe that, at the interpolation points, the variance decreases drastically, reaching s^2 values in each of them (except for errors of the order of 10^{-4}): this is the behavior we expect in the presence of observations subject to noise. This behavior is more pronounced in the case of exponential kernels.
- In addition, it can be seen that the variance increases at the edges of the domain: this is typical behavior of Gaussian process regression.

- Finally, the exponential squared kernel causes the mean of the predictions to be a much smoother function: in fact, the contour lines are more regular than those for the exponential kernel.

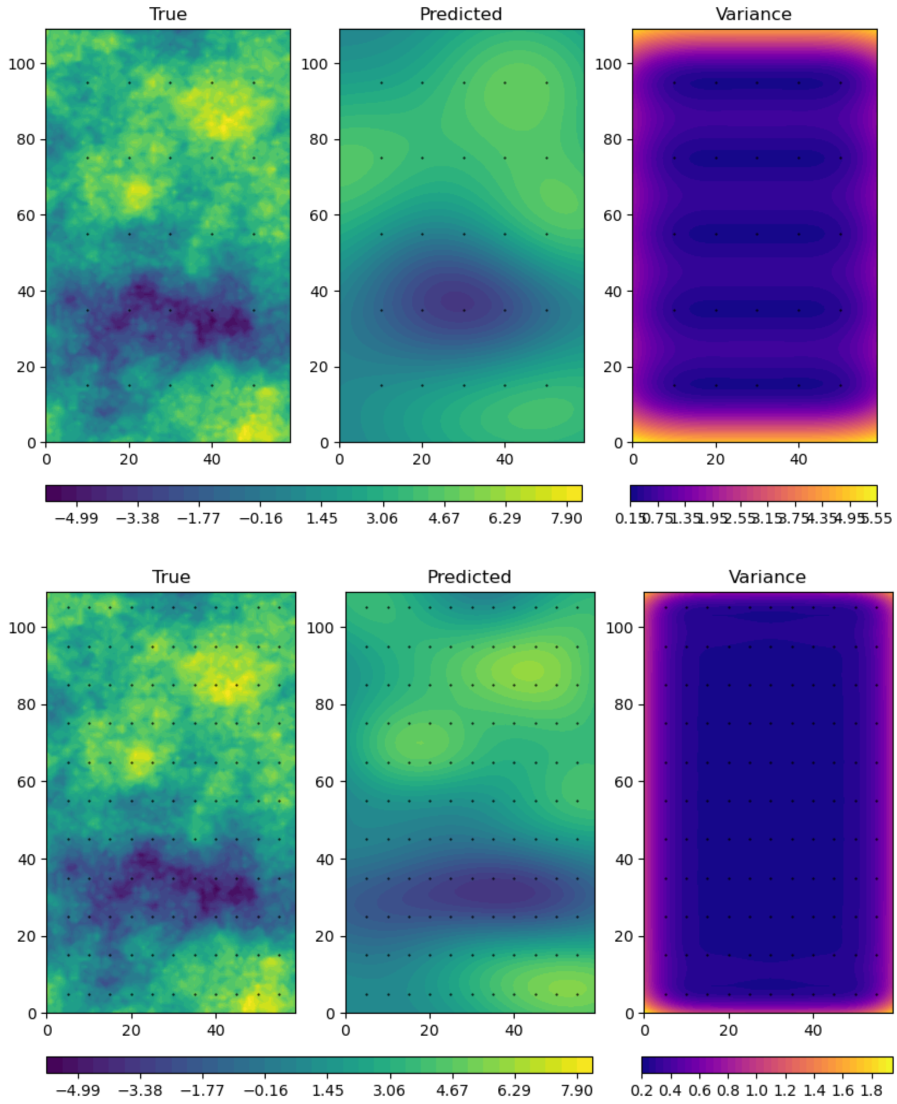

FIGURE 6

2D Gaussian Process: Exponential Kernel

4.4 GENERATE PROCESSES: CIRCULANT EMBEDDING

To generate some realization of the 2D conditional Gaussian random fields $\mathbf{f}(Y)|\tilde{\mathbf{f}}(Z) \sim \mathcal{N}(\tilde{\mathbf{m}}(Y \mid Z), \mathbf{K}(Y|Z))$, there are several methods that can be applied. The standard methods involve factorizing the covariance matrix $\mathbf{K}(Y|Z)$ by means of Cholesky's method or SVD: this approach, however, may be computationally heavy when this matrix is large. For this reason we decide to resort to Circulant Embedding, that is, the generation of a circular matrix from the covariance matrix $\mathbf{K}(Y|Z)$, which enjoys convenient algebraic properties. It is in fact easily factorized by means of the FFT since its eigenvalues and eigenvectors are computable by known formulas. Once the circular matrix $\mathbf{P}$ has been generated, the information needed to generate a Gaussian process can be derived from it ([3], Chapter 3.3.2). However, the construction of the matrix $\mathbf{P}$ in 2D is not as straightforward as in the one-dimensional case: the notion of homogeneity and isotropy differ as soon as we move to dimensions greater than 1. In light of this difference, we find that the covariance matrix $\mathbf{K}(Y|Z)$ of the conditional process is not homogeneous for both kernels: it is therefore not possible to proceed by deriving the circular matrix directly from it. Consequently, we choose to apply circulant embedding to the matrix $\mathbf{K}(Y)$ to obtain a vector of Gaussian random variables, using Agorithm 3.

Algorithm 3 Circulant Embedding 2D

```
Build $\mathbf{P}$ starting from $\mathbf{K}(Y)$
Calculate $\mathbf{W} = \text{fft}(\mathbf{P})$, the 2-dimensional FFT of $\mathbf{P}$
Check that all elements of $\mathbf{W}$ are positive
Generate matrix $\mathbf{X}$ with size of $\mathbf{W}$ containing i.i.d. normal variables
Calculate $\tilde{\mathbf{Z}} = \text{ifft}(\sqrt{\mathbf{W}} \odot \mathbf{X})$, the 2-dimensional inverse FFT of $\sqrt{\mathbf{W}} \odot \mathbf{X}$
Generate $\mathbf{f}(Y) = Re(\tilde{\mathbf{Z}})|_Y + Im(\tilde{\mathbf{Z}})|_Y$
return $\mathbf{f}(Y)$
```

We observe that, within the 6600 values just generated, there are also the sampling points, whose true value is known. In light of this, "a posteriori" conditioning can be applied to the observations, using Algorithm 4.

Algorithm 4 Generation from conditional Gaussian distribution

```
Generate $\mathbf{f}(Y) \sim \mathcal{N}(\mathbf{0}, \mathbf{K}(Y))$ using 2D Circulant Embedding
Extract indexes of points $Z$ and evaluate $\mathbf{f}_{\mathbf{new}}(Z) = \mathbf{f}(Y)|_Z$
Generate $\mathbf{f}(Y)|\mathbf{f}(Z) = \mathbf{f}(Y) + \mathbf{K}(Y,Z)\mathbf{K}(Z)^{-1}(\mathbf{f}_{\mathbf{new}}(Z) - \mathbf{f}(Z))$
return $\mathbf{f}(Y)|\mathbf{f}(Z)$
```

As a result, we obtain the conditional Gaussian processes $\mathbf{f}(Y)|\tilde{\mathbf{f}}(Z) \sim \mathcal{N}(\tilde{\mathbf{m}}(Y|Z), \mathbf{K}(Y|Z))$, whose visual representation follows.

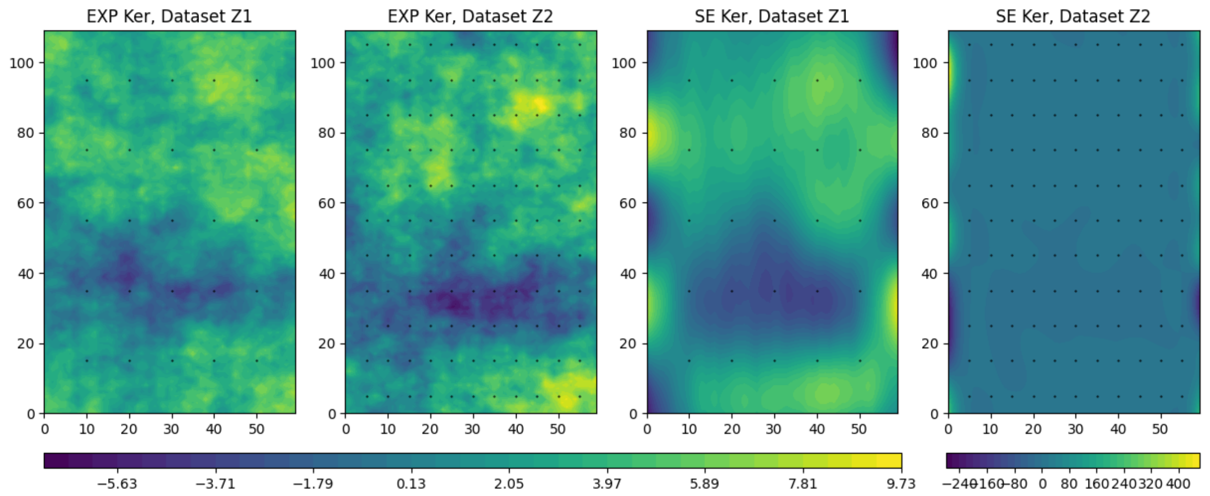

FIGURE 7
Gaussian Processed obtained via Circular Embedding technique

Observe that this procedure is extraordinarily unstable for the SE kernel when we deal with the second set of training points Z2.

4.5 Monte Carlo estimator

At this stage, our aim is to determine whether the permeability at certain coordinates exceeds a predetermined threshold. Specifically, we are examining four critical locations with coordinates $x =$

$35, 40, 45, 50$ and $y = 85$ (and corresponding indexes $5135, 5140, 5145, 5150$). We want to estimate $\mathbb{P}(\max_{i=1,\dots,4} f(x_i, y_i) \geq 8) = \mathbb{E}(\psi(\mathbf{x}, \mathbf{y}))$, where $\psi(\mathbf{x}, \mathbf{y}) = \mathbf{1}_{\max_{i=1,\dots,4} f(x_i,y_i) \geq 8}$.

Theorem 1 *Let $\mathbf{X}$ be a Gaussian vector of size n with mean $\boldsymbol{\mu}$ and covariance matrix Σ. Then the linear transformation $\mathbf{Y} = A\mathbf{X} + \mathbf{b}$ will transform $\mathbf{X}$ into a Gaussian vector $\mathbf{Y}$ with mean $A\boldsymbol{\mu} + \mathbf{b}$ and covariance matrix $A\Sigma A^T$.*

The result of this theorem is that generating a Gaussian process limited to the four points of interest will always produce a Gaussian process whose mean and covariance are obtained evaluating the original one in those points. Therefore, we can use the Monte Carlo method (CMC) to generate a large number of processes and evaluate the probability with significantly lower computational cost. Using this method, we can estimate the probability and the $1 - \alpha$ confidence interval.

Given a tolerance for the range of the confidence interval, the number N of the simulations to perform is chosen with a two stages Monte Carlo method, whose algorithm is described in [[2], Algorithm 5.1].

The results obtained can be found in Table (3):

Z2	$\bar{N}$	$\hat{\mu}_{\bar{N}}$	$\hat{\sigma}_{\bar{N}}$	tol
EXP	9360182	0.024976	0.15605	1e-4
SE	2633285	6.7596e-5	0.00822	1e-5

Z2	$\bar{N}$	$\hat{\mu}_{\bar{N}}$	$\hat{\sigma}_{\bar{N}}$	tol
EXP	26223987	0.073693	0.26127	1e-4
SE	768284	2.6032e-5	0.00161	1e-5

TABLE 3
Estimation with Two stages MC for datasets Z1 and Z2

We notice that the simulations carried out using the EXP kernel tend to produce higher results than those using the SE kernel. This is to be expected, as the mean values at specific points with the EXP kernel are larger than those with the SE kernel, even though they are all less than 8. Additionally, the variance is also higher for the EXP kernel, which means there is a higher probability of reaching the value of 8 due to a larger deviation from the mean. In contrast, this is a less common occurrence when using the SE kernel.

4.6 VARIANCE REDUCTION

One of the major drawbacks of the Crude Monte Carlo (CMC) method is its slow convergence rate of $O(\frac{1}{\sqrt{N}})$, resulting in a lengthy process to obtain precise results. One method to decrease the variance of Monte Carlo and consequently accelerating convergence is the use of Antithetic Variables (AV). AV involves using a probability distribution that is symmetrical about the origin to generate the random negatively correlated pairs of samples used in the simulation. This technique helps to reduce the approximation error and achieve more accurate results in a shorter amount of time.

Theorem 2 *Assume that the random variable Z has the expression $Z = \psi(X)$, with $X = (X_1, \dots, X_d)$ a random vector with independent components, such that*

- *X has a symmetric distribution around its mean, i.e. $2\mathbb{E}[X] - X \sim X$*
- *ψ is a monotone function in each of its arguments.*

Then $Z = \psi(X)$ and $Z_a = \psi(2\mathbb{E}[X] - X)$ satisfy $\mathbb{E}[Z] = \mathbb{E}[Z_a]$ *and* $\mathrm{Cov}(Z, Z_a) < 0$.

It can be easily confirmed that the conditions required for this case are satisfied. Specifically, $\psi(\mathbf{X})$ is a characteristic function and so monotone with respect to each variable in the random vector $\mathbf{X} = (X_1, \cdots, X_4)$. Additionally, since $\mathbf{X}$ is a Gaussian random vector, its distribution is symmetric around its mean $\boldsymbol{\mu}(X)$. A Monte Carlo approximation with antithetic variables can be constructed using [[2], Algorithm 6.1]. We generate different estimation of CMC and AV increasing N. In Table 4, we show the confidence interval for the last observations ($N = 1e8$).

Z1	$\hat{\mu}_{CMC}$	$\hat{\mu}_{AV}$	Z2	$\hat{\mu}_{CMC}$	$\hat{\mu}_{AV}$
EXP	0.0249 ± 9.662e-5	0.0250 ± 9.560e-5	EXP	0.0737 ± 1.6193e-4	0.0737 ± 1.5539e-4
SE	7.4e-5 ± 5.331e-6	7.23e-5 ± 5.269e-6	SE	3.7e-6 ± 1.1922e-6	3.7e-6 ± 1.1922e-6

TABLE 4
Confidence interval of CMC and AV for datasets Z1 and Z2

In Figure (8), we plot the expected values predicted and the confidence intervals with respect to N, both for CMC and AV. The use of the antithetic variables method slightly decreases the confidence interval for the expected value compared to the crude Monte Carlo method, as demonstrated by our assumptions. However, when using the SE kernel with small sample sizes N, the estimate of the expected value becomes highly variable and may even be zero. To obtain a more accurate estimate, a larger sample size is required, which increases computational cost. The results show that the AV method is more effective when using the EXP kernel, rather than for modeling less probable events like the SE kernel.

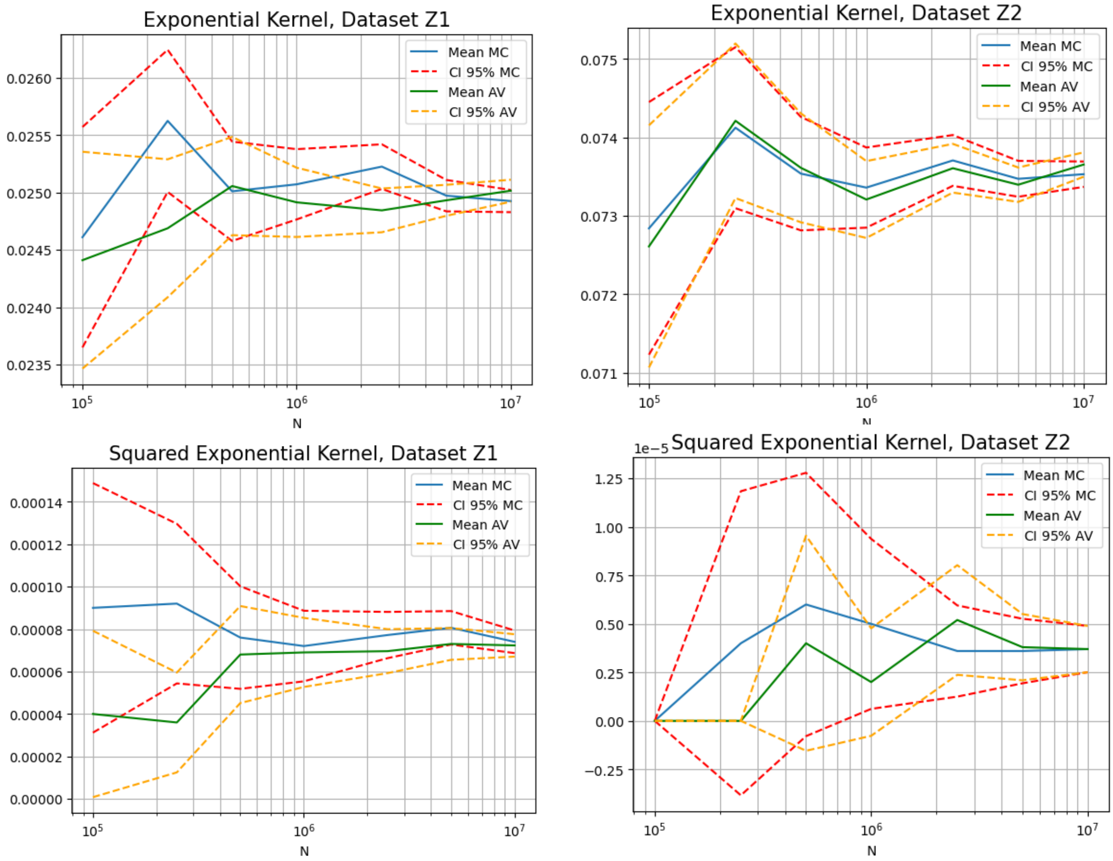

FIGURE 8
Monte Carlo estimations and Antithetic Variables to reduce variance

5 CONCLUSION

The use of Gaussian Process Regression in this project has allowed us to successfully reconstruct a one-dimensional function (Section 3) and estimate the permeability field through two-dimensional observations in a geoscience problem (Section 4). However, we have observed that the choice of the kernel can significantly impact the performance of the process. This aligns with the principles of the Bayesian approach, in which the prior distribution carries significant weight. We showed the utility of Gaussian Process Regression in geoscience applications, but also emphasized the need to carefully consider the kernel selection in order to achieve optimal results. The implementation of the algorithms to carry out the project can be found in https://github.com/francpp/Gaussian-Process-Regression.

A Appendix: Simulations

A.1 Task1: Recovering a simple function

```
# define initial parameters
NNz = [10, 100]
Ny = 1000
var = 0.1
s_2 = 0.01
ll = [0.1, 0.3, 1.0]
f_tilde = lambda z: np.sin(z)

# do simulations for all parameters
for ker in range(2):
    if ker == 0:
        kernel = lambda x1, x2, l, var: kernel_exp(x1,x2,l,var)
        type_ker = ’ker_exp’
    elif ker == 1:
        kernel = lambda x1, x2, l, var: kernel_SE(x1,x2,l,var)
        type_ker = ’ker_SE’

    for l in ll:
        for Nz in NNz:

            #create domain
            Z, f_tilde_Z, Y = create_Domain1(Nz, f_tilde, Ny)
            #evaluate kernels
            K_Z, K_Y, K_ZY = evaluate_kernels(Z, Y, kernel, l, var)
            #computes conditional mean and variance of the process
            mu_YgivenZ, K_YgivenZ = prediction(f_tilde_Z, K_Z, K_Y, K_ZY, s_2)
            #samples again
            y_cond=g_process(mu_YgivenZ, K_YgivenZ, n_process = 3)

# Minimize THETA of the log-likelihood

Z0, f_tilde_Z0, Y = create_Domain1(NNz[0], f_tilde, Ny)
res_exp0 = minimize(minimizer(Z0, f_tilde_Z0, 0, prob = 1), [1, 0.1, 0.01],
               bounds=((0.01, None), (0, None), (0, None)),
               method=’L-BFGS-B’)
res_SE0 = minimize(minimizer(Z0, f_tilde_Z0, 1, prob = 1), [1, 0.1, 0.001],
                  bounds=((0, None), (0, None), (0, None)), tol = 1e-1,
                  method=’L-BFGS-B’)

Z1, f_tilde_Z1, Y = create_Domain1(NNz[1], f_tilde, Ny)
res_exp1 = minimize(minimizer(Z1, f_tilde_Z1, 0, prob = 1), [1, 0.1, 0.01],
               bounds=((0.01, None), (0, None), (0, None)),
               method=’L-BFGS-B’)
res_SE1 = minimize(minimizer(Z1, f_tilde_Z1, 1, prob = 1), [1, 0.1, 0.001],
                  bounds=((0, None), (0, None), (0, None)), tol = 1e-1,
                  method=’L-BFGS-B’) # occhio ai parametri iniziali

# Compute a run with the optimal parameters
for KER in range(0,2):
    for PTS in range(0,2):

        #define the case study
        kernel, Z, f_tilde_Z, l_opt, var_opt, s_2_opt = case_study1(KER, PTS, Z0,
    f_tilde_Z0, Z1, f_tilde_Z1, res_exp0, res_SE0, res_exp1, res_SE1)

        #evaluate kernels
        K_Z, K_Y, K_ZY = evaluate_kernels(Z, Y, kernel, l_opt, var_opt)
        #computes conditional mean and variance of the process
        mu_YgivenZ, K_YgivenZ = prediction(f_tilde_Z, K_Z, K_Y, K_ZY, s_2_opt)
        #samples again
        y_cond=g_process(mu_YgivenZ, K_YgivenZ, n_process = 3)
```

A.2 TASK2: GP REGRESSION ON A PERMABILITY FIELD

```
# load files and extract values
perm = np.load('true_perm.npy')

# define the coordinates of available observations
Coord1 = (np.array([np.linspace(15, 95, 5), np.linspace(10, 50, 5)]).T).astype(int)
Coord2 = (np.array([np.linspace(5, 105, 11), np.linspace(5, 55, 11)]).T).astype(int)

# define the domains
Mesh_total, Points_total = create_Domain2(perm)
Mesh1, Values1, Points1 = create_Dataset(perm, Coord1)
Mesh2, Values2, Points2 = create_Dataset(perm, Coord2)

# minimize the log-likelihood for Points1
res1_exp = minimize(minimizer(Points1, Values1, ker = 0, prob=2),
                x0 = [1, 0.1, 0.1],
                bounds=((1e-10, None), (1e-10, None), (1e-10, None)),
                method='L-BFGS-B')
res1_se = minimize(minimizer(Points1, Values1, ker = 1, prob=2),
                 x0 = [5., 8, 1., 1.],
                 bounds=((1e-10, None), (1e-10, None), (1e-10, None),(1e-10, None)),
                 method='L-BFGS-B')

# minimize the log-likelihood for Points2
res2_exp = minimize(minimizer(Points2, Values2, ker = 0, prob=2),
                x0 = [1, 0.1, 0.1],
                bounds=((1e-10, None), (1e-10, None), (1e-10, None)),
                method='L-BFGS-B')
res2_se = minimize(minimizer(Points2, Values2, ker = 1, prob=2),
                 x0 = [5., 10., 1., 1.],
                 bounds=((1e-10, None), (1e-10, None), (1e-10, None),(1e-10, None)),
                 method='L-BFGS-B')

# Compute a run with the optimal parameters

for KER in range(2):
    for PTS in range(1,3):
        kernel, Points, Values, l_opt, var_opt, s_2_opt = case_study2(KER, PTS,
    Points1, Values1, Points2, Values2, res1_exp, res1_se, res2_exp, res2_se)

        # Fit Gaussian Process Regression
        K_Z, K_Y, K_ZY = evaluate_kernels(Points, Points_total, kernel, l = l_opt,
    var = var_opt)
        mean_perm, cov_perm = prediction(Values, K_Z, K_Y, K_ZY, s_2 = s_2_opt)
        var_perm = np.diag(cov_perm)

        # Generate gaussian processes with Circular Embedding
        perm_cond = conditional(K_ZY, K_Z, K_Y, Points, Values, n_process = 1000)

        # Evaluate the expected value of psi with Monte Carlo and Antithetic
    Variables
        Critical = (np.array([np.ones(4)*85, np.array([35, 40, 45, 50])]).T).astype(
    int)
        indexes = np.array([60*Critical[0,0] + i for i in Critical[:,1]])
        mean_interest = mean_perm[indexes]
        var_interest = cov_perm[indexes,:][:,indexes]
        N, mean_est, std_est = MC_2stages(mean_interest, var_interest, tol = tol, N0 = N0)

        # Generate additional data for the convergence graph
        Ns_all = np.array([100000, 250000, 500000, 1000000, 2500000, 5000000, 10000000])
        means_MC, stds_MC, means_AV, stds_AV = plot_MC_AV(Ns_all, mean_interest, var_interest, KER, PTS)
```

B APPENDIX: HELPERS

B.1 KERNELS

```
# define euclidean-distance between m d-dim points
def dist_2(X1, X2):
    rows, cols = np.indices((X1.shape[0], X2.shape[0]))
    distances = np.sqrt(np.sum((X1[rows, :] - X2[cols, :])**2, axis=2))
    return distances

# define weighted-distance between m d-dim points
def dist_w(X1, X2, l):
    rows, cols = np.indices((X1.shape[0], X2.shape[0]))
    distances = np.sum(((X1[rows, :] - X2[cols, :])/l)**2, axis=2)
    return distances

# define exponential kernel
def kernel_exp(X1, X2, l, var):
    distances = dist_2(X1, X2)
    return var * np.exp(-distances/l)

# define square exponential kernel
def kernel_SE(X1, X2, l, var):
    distances = dist_w(X1, X2, l)
    return var * np.exp(-0.5*distances)

# evaluate kernels given the dataset and hyperparameters
def evaluate_kernels(Z, Y, kernel, l, var):
    K_Z = kernel(Z,Z,l,var)
    K_Y = kernel(Y,Y,l,var)
    K_ZY = kernel(Z,Y,l,var)
    return K_Z, K_Y, K_ZY
```

B.2 DOMAINS

```
# create Dataset for task 1
def create_Domain1(Nz, f_tilde, Ny):
    Z = 2*np.pi*np.random.uniform(size = Nz)
    f_tilde_Z = f_tilde(Z)
    Y = np.linspace(0, 2*np.pi, Ny).reshape(-1, 1)
    return Z, f_tilde_Z, Y

# create datasets for task2
def create_Domain2(perm):
    Y = perm.shape[0]
    X = perm.shape[1]
    Mesh_total = np.meshgrid(np.arange(Y), np.arange(X))
    Points_total = np.array([Mesh_total[0].T.flatten(), Mesh_total[1].T.flatten()]).T
    return Mesh_total, Points_total

def create_Dataset(perm, Coord):
    Mesh = np.meshgrid(Coord[:,0], Coord[:,1])
    Points = np.array([Mesh[0].T.flatten(), Mesh[1].T.flatten()]).T
    Values = perm[Points[:, 0], Points[:,1]]
    return Mesh, Values, Points
```

B.3 Optimization of hyperparameters

```
def minimizer(X, Y, ker, prob):
    if ker == 0:
        kernel = lambda x1, x2, l, var: kernel_exp(x1,x2,l,var)
    elif ker == 1:
        if prob == 1:
            kernel = lambda x1, x2, l, var: kernel_SE(x1,x2,l,var)
        elif prob == 2:
            kernel = lambda x1, x2, l1, l2, var: kernel_SE(x1,x2,np.array([l1,l2]),var)
    def log_likelihood(theta):
        if (prob == 2 and ker == 1):
            K = kernel(X, X, l1=theta[0], l2=theta[1], var=theta[2]) + theta[3] * np.eye(len(X))
        else:
            K = kernel(X, X, l=theta[0], var=theta[1]) + theta[2] * np.eye(len(X))

        L = np.linalg.cholesky(K)
        U = solve_triangular(L, Y, lower=True)
        B = solve_triangular(L.T, U, lower=False)
        logL = np.sum(np.log(np.diagonal(L))) + 0.5 * Y.dot(B)
        return logL

    return log_likelihood
```

B.4 Sampling of processes

```
# predict mean and covariance of the conditioned gaussian process
def prediction(f, K_Z, K_Y, K_ZY, s_2):
    I = s_2*np.eye(K_Z.shape[0])
    L = np.linalg.cholesky(K_Z+I)
    mu_pred = K_ZY.T@(np.linalg.solve(L.T, np.linalg.solve(L, f)))
    cov_pred = K_Y - K_ZY.T@(np.linalg.solve(L.T, np.linalg.solve(L, K_ZY)))
    return mu_pred, cov_pred

# generate different gaussian processes
def g_process(mean, sigma, n_process = 1):
    U,D,_ = np.linalg.svd(sigma)
    A = U@np.diag(np.sqrt(D))
    Ny = sigma.shape[0]
    proc = mean + A@np.random.standard_normal(size = (Ny,n_process))
    return proc

# Circulant Embedding in 2D
def build_P(sigma):
    upper_left_P = np.empty((60,110))
    for j in range(6600):
        delta_y = int(j/60)
        delta_x = int(j%60)
    upper_left_P[delta_x, delta_y] = sigma[0,j]
    upper_right_P = np.flip(upper_left_P[:,1:], axis=1)
    upper_P = np.c_[upper_left_P, upper_right_P]
    lower_P = np.flip(upper_P[1:,:], axis=0)
    P = np.r_[upper_P, lower_P]
    return P

def CE(sigma, n_process):
    P = build_P(sigma)
    W = fft2(P)
    gauss_process = np.zeros((6600, n_process))
    for i in range(n_process):
        Xi = np.random.standard_normal(size=W.shape)
        T = 2*np.sqrt(59*109*W)*Xi
        Z = ifft2(T)
        Z_prime = Z.real + Z.imag
        sample = (Z_prime[:60, :110]).T.flatten()
        gauss_process[:,i] = sample
    return gauss_process

# condition the zero-mean gaussian processes to the observations
def conditional(K_ZY, K_Z, K_Y, Points, Values, n_process):
    index = 60*Points[:,0]+Points[:,1] #indexes of Z
    precond = CE(K_Y, n_process)
    z_iniz = precond[index]
    post = precond + K_ZY.T@np.linalg.solve(K_Z, Values.reshape(-1,1)-z_iniz)
    return post
```

B.5 Monte Carlo and Variance Reduction

```
# define the function whose expected value we want to calculate
def csi(ip):
    cnt = np.count_nonzero(ip>8, axis=0)
    cnt[cnt>0]=1
    return cnt

# define the Crude Monte Carlo method
def MC(mu, sigma, N):
    ip = g_process(mu, sigma, n_process = N)
    est = csi(ip)
    mean_est = np.mean(est)
    std_est = np.std(est, ddof = 1)
    return mean_est, std_est

# define the 2stages Monte Carlo method
def MC_2stages(mu, sigma, tol, N0 = 100000):
    alpha = 0.05
    c = norm.ppf(1-alpha/2) #1.96
    N_old = N0
    mean_est_, std_est_ = MC(mu, sigma, N_old)
    N = N_old
    if(std_est_!=0):
        N = int((c*std_est_/tol)**2)
    mean_est, std_est = MC(mu, sigma, N)
    while std_est >= std_est_:
        N_old = N
        mean_est_, std_est_ = MC(mu, sigma, N_old)
        if std_est_ != 0:
            N = int((c*std_est_/tol)**2)
        mean_est, std_est = MC(mu, sigma, N)
    return N, mean_est, std_est

# define the antithetic variables method
def MC_AV(mu, sigma, N):
    N2 = int(N/2)
    X = g_process(mu, sigma, n_process = N2)
    Xa = 2*mu - X
    Z = csi(X)
    Za = csi(Xa)
    est = (Z + Za)/2
    mean_est = np.mean(est)
    std_est = np.std(est, ddof=1)
    return mean_est, std_est

# compute different simulations for MC and AV
def plot_MC_AV(Ns_all, mean_interest, var_interest, KER, PTS):
    for i in range(len(Ns_all)):
        means_MC[i], stds_MC[i] = MC(mean_interest, var_interest, Ns_all[i])
        means_AV[i], stds_AV[i] = MC_AV(mean_interest, var_interest, Ns_all[i])
    upperlim = means_MC + 1.96*stds_MC/np.sqrt(Ns_all)
    lowerlim = means_MC - 1.96*stds_MC/np.sqrt(Ns_all)
    upperlim = means_AV + 1.96*stds_AV/np.sqrt(Ns_all/2)
    lowerlim = means_AV - 1.96*stds_AV/np.sqrt(Ns_all/2)
    return means_MC, stds_MC, means_AV, stds_AV
```

C APPENDIX: GRADIENTS OF THE MARGINAL LOG-LIKELIHOOD

Before proceeding to calculate the marginal log-likelihood gradients analytically, we recall that, given a square matrix $A(\mathbf{x})$, the derivative of the matrix and the determinant result, respectively:

$$\frac{\partial A^{-1}}{\partial x_i} = -A^{-1}\frac{\partial A}{\partial x_i}A^{-1} \qquad \text{and} \qquad \frac{\partial \log|\det(A)|}{\partial x_i} = \operatorname{Tr}\left(A^{-1}\frac{\partial A}{\partial x_i}\right)$$

Furthermore, for the sake of ease of notation we introduce $\tilde{\mathbf{K}} = \mathbf{K}(Z) + s^2\mathbf{I}$ and proceed with the analytical calculation:

- Notice that $\dfrac{\partial \tilde{\mathbf{K}}}{\partial s^2} = \mathbf{I}$, so:

$$\begin{aligned}\frac{\partial L}{\partial s^2} &= -\frac{1}{2}v^T\left[-\tilde{\mathbf{K}}^{-1}\left(\frac{\partial \tilde{\mathbf{K}}}{\partial s^2}\right)\tilde{\mathbf{K}}^{-1}\right]v - \frac{1}{2}\frac{\partial}{\partial s^2}\left[\log|\det(\tilde{\mathbf{K}})|\right] = \\ &= \frac{1}{2}v^T\left[\tilde{\mathbf{K}}^{-1}\tilde{\mathbf{K}}^{-1}\right]v - \frac{1}{2}\frac{1}{\det(\tilde{\mathbf{K}})}\frac{\partial \det(\tilde{\mathbf{K}})}{\partial s^2} = \\ &= \frac{1}{2}v^T\left[\tilde{\mathbf{K}}^{-1}\tilde{\mathbf{K}}^{-1}\right]v - \frac{1}{2}\operatorname{Tr}\left(\tilde{\mathbf{K}}^{-1}\frac{\partial \tilde{\mathbf{K}}^{-1}}{\partial s^2}\right) = \\ &= \frac{1}{2}v^T\left[\tilde{\mathbf{K}}^{-1}\tilde{\mathbf{K}}^{-1}\right]v - \frac{1}{2}\operatorname{Tr}\left(\tilde{\mathbf{K}}^{-1}\right)\end{aligned}$$

- one observes that $\dfrac{\partial \tilde{\mathbf{K}}}{\partial \sigma^2} = \dfrac{\partial \mathbf{K}}{\partial \sigma^2} = \mathbf{K}_{\sigma^2} = \begin{cases} \exp\left\{-\frac{\|\mathbf{x}\text{ - }\mathbf{x'}\|_2}{l}\right\} & (exp) \\ \exp\left\{-\frac{1}{2}\sum_{i=1}^{k}\frac{\left(x_i - x_i'\right)^2}{\ell_i^2}\right\} & (se) \end{cases}$

$$\begin{aligned}\frac{\partial L}{\partial \sigma^2} &= -\frac{1}{2} v^T \left[-\tilde{\mathbf{K}}^{-1} \left(\frac{\partial \tilde{\mathbf{K}}}{\partial \sigma^2} \right) \tilde{\mathbf{K}}^{-1} \right] v - \frac{1}{2} \frac{\partial}{\partial \sigma^2} \left[\log |\det(\tilde{\mathbf{K}})| \right] = \\ &= \frac{1}{2} v^T \left[\tilde{\mathbf{K}}^{-1} \mathbf{K}_{\sigma^2} \tilde{\mathbf{K}}^{-1} \right] v - \frac{1}{2} \frac{1}{\det(\tilde{\mathbf{K}})} \frac{\partial \det(\tilde{\mathbf{K}})}{\partial \sigma^2} = \\ &= \frac{1}{2} v^T \left[\tilde{\mathbf{K}}^{-1} \mathbf{K}_{\sigma^2} \tilde{\mathbf{K}}^{-1} \right] v - \frac{1}{2} \operatorname{Tr} \left(\tilde{\mathbf{K}}^{-1} \frac{\partial \tilde{\mathbf{K}}^{-1}}{\partial \sigma^2} \right) = \\ &= \frac{1}{2} v^T \left[\tilde{\mathbf{K}}^{-1} \mathbf{K}_{\sigma^2} \tilde{\mathbf{K}}^{-1} \right] v - \frac{1}{2} \operatorname{Tr} \left(\tilde{\mathbf{K}}^{-1} \mathbf{K}_{\sigma^2} \right)\end{aligned}$$

- notice that $\frac{\partial \tilde{\mathbf{K}}}{\partial \ell_i} = \frac{\partial \mathbf{K}}{\partial \ell_i} = \mathbf{K}_{\ell_i} = \begin{cases} \frac{1}{\ell_i^2} \mathbf{K} & (exp) \\ \frac{1}{\ell_i^3} \mathbf{K} & (se) \end{cases}$

$$\begin{aligned}\frac{\partial L}{\partial \ell_i} &= -\frac{1}{2} v^T \left[-\tilde{\mathbf{K}}^{-1} \left(\frac{\partial \tilde{\mathbf{K}}}{\partial \ell_i} \right) \tilde{\mathbf{K}}^{-1} \right] v - \frac{1}{2} \frac{\partial}{\partial \ell_i} \left[\log |\det(\tilde{\mathbf{K}})| \right] = \\ &= \frac{1}{2} v^T \left[\tilde{\mathbf{K}}^{-1} \mathbf{K}_{\ell_i} \tilde{\mathbf{K}}^{-1} \right] v - \frac{1}{2} \frac{1}{\det(\tilde{\mathbf{K}})} \frac{\partial \det(\tilde{\mathbf{K}})}{\partial \ell_i} = \\ &= \frac{1}{2} v^T \left[\tilde{\mathbf{K}}^{-1} \mathbf{K}_{\ell_i} \tilde{\mathbf{K}}^{-1} \right] v - \frac{1}{2} \operatorname{Tr} \left(\tilde{\mathbf{K}}^{-1} \frac{\partial \tilde{\mathbf{K}}^{-1}}{\partial \ell_i} \right) = \\ &= \frac{1}{2} v^T \left[\tilde{\mathbf{K}}^{-1} \mathbf{K}_{\ell_i} \tilde{\mathbf{K}}^{-1} \right] v - \frac{1}{2} \operatorname{Tr} \left(\tilde{\mathbf{K}}^{-1} \mathbf{K}_{\ell_i} \right)\end{aligned}$$

References

[1] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*. Vol. 2. 3. MIT press Cambridge, MA, 2006.

[2] Fabio Nobile. *Stochastic Simulation: Lecture Notes*. 2022.

[3] Andreas Van Barel. ‘Multilevel Monte Carlo Methods for Robust Optimization of Partial Differential Equations’. In: (2021).