IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Gaussian Processes for Optimal Sensor Position

Background & Progress Report

Author:
Adrian LÖWENSTEIN

Supervisor:
Rossella ARCUCCI
Miguel MOLINA-SOLANA

Submitted in partial fulfillment of the requirements for the MSc degree in Computing (Machine Learning) of Imperial College London

June 2019

Contents

Chapter 1

Project Proposal

Project Proposal:

Gaussian processes (GP) have been widely used since the 1970s in the fields of geostatistics and meteorology. Current applications are in diverse fields including sensor placement In this project, we propose the employment of a GP model to calculate the optimal spatial positioning of sensors to study and collect air pollution data in big cities. We will then validate the results by means of a data assimilation software with the data at the proposed positions.

Dataset:

London South Bank University (LSBU) air pollution data (velocity, tracer)

Chapter 2

Background

In this chapter, we will cover the literature and the theory that will be used throughout the project. First, we will review the context of the project and how it fits into the **MAGIC** project. Then our focus goes to the definition of **Gaussian Processes** (GP) and how they are used in the context of geospatial data. Furthermore, the use of GP relies heavily on **Covariance** matrixes which need to be estimated. Those tools enable us to create **optimisation** algorithms for the position of sensors. Finally, we will quickly explore the concepts of **Data Assimilation** (DA) that will be used to validate the results of the optimisation.

2.1 The MAGIC Project

This work is done in the context of the **Managing Air for Green Inner Cities** project. This is a multidisciplinary project and has for objective to find solutions to the pollution and heating phenomenons in cities. Traditionally, urban environmental control relies on polluting and energy consuming heating, ventilation and cooling (HVAC) systems. The usage of the systems increases the heat and the pollution levels, inducing an increased need for the HVAC. The MAGIC project aims at breaking this vicious circle and has for objective to provide tools to make possible the design of cities acting as a natural HVAC system.

This has been extensively discussed by Song et al. (2018). For this purpose, integrated management and the decision-support system is under development. It includes a variety of simulations for pollutants and temperature at different scales; a set of mathematical tools to allow fast computation in the context of real-time analysis; and cost-benefit models to asses the viability of the planning options and decisions.

As explained by Song et al. (2018), the test site which has been selected to conduct the study is a real urban area located in London South Bank University (LSBU) in Elephant and Castle, London. In order to investigate the effect of ventilation on the cities problem, researchers in the MAGIC project have created simulations and experiments both in outdoor and indoor conditions, on the test site. They used wind tunnel experiments and computational fluid dynamics (CFD) to simulate the outdoor environment. Further works include the development of reduced-order modelling (ROM) in order to make faster the simulations while keeping a high level of accuracy (Arcucci et al., 2018).

Another key research direction in the use Data Assimilation (DA) and more specifically Variational DA (VarDA) for assimilating measured data in real time and allowing better prediction of the model in the near future (Arcucci et al., 2018). The further use of those methods would be the optimisation of the position of the sensors which provide information for the VarDA.

2.2 Gaussian Processes

In this chapter, we will review Gaussian Processes (GP) which are probabilistic models for spatial predictions based on observations assumption.

As explained by Rasmussen and Williams (2006, p. 29), the history of Gaussian Processes goes back at least as far as the 1940s. A lot of usages were developed in various fields. Notably for predictions in spatial statistics (Cressie, 1991). Applied in particular in Geostatistics with methods known as *kringing*, and in Meteorology. Gradually GP started to be used in more general cases for regression. Nowadays it is often used in the context of Machine Learning.

For our problem, we will be modelling the data of the sensor with GPs.

2.2.1 Sensor Data Modelling

2.2.1.1 Multivariate Gaussian Distribution

GP will serve as a basic tool in our project. In the space we are monitoring we have a certain number of sensors measuring a certain quantity, such as temperature, pressure, speed of the wind or the concentration of a pollutant at a given position. We assume that the measured quantity has a *multivariate Gaussian joint distribution* between each point of the space. The associated random variable is $\mathcal{X}_{\mathcal{V}}$ for the set of locations $\mathcal{V}$ we would have the following distribution : $P(\mathcal{X}_{\mathcal{V}} = \boldsymbol{x}_{\mathcal{V}}) \sim \mathcal{N}(\mu_{\mathcal{V}}, K_{\mathcal{V}\mathcal{V}})$, or explicitly :

$$P(\mathcal{X}_{\mathcal{V}} = \boldsymbol{x}_{\mathcal{V}}) \frac{1}{(2\pi)^{n/2}|K_{\mathcal{V}\mathcal{V}}|} \exp^{-\frac{1}{2}(\boldsymbol{x}_{\mathcal{V}}-\mu_{\mathcal{V}})^T K_{\mathcal{V}\mathcal{V}}^{-1}(\boldsymbol{x}_{\mathcal{V}}-\mu_{\mathcal{V}})} \tag{2.1}$$

2.2.1.2 Prediction with Gaussian Processes

Let us still consider that we have the set of locations $\mathcal{V}$ and a set of sensors $\mathcal{A}$. In order to predict the quantity at positions were we have no sensors ($\mathcal{V}\backslash\mathcal{A}$) we can use a Gaussian Process. This allows us to infer a function belonging the the function space

: $\mathcal{GP}(\mathcal{M}(\boldsymbol{x}), \mathcal{K}(\boldsymbol{x}, \boldsymbol{x}'))$. This GP is associated with a **mean function** $\mathcal{M}(\cdot)$ and a symmetric positive-definite **kernel function** $\mathcal{K}(\cdot, \cdot)$. We will denote the mean function values for a set of positions $\mathcal{A}$ by $\mu_{\mathcal{A}}$ and the kernel function values, or covariance matrix, between those points by $K_{\mathcal{A}}$. More detailed definitions are available in Rasmussen and Williams (2006, p. 13-16).

For a set of observations $\boldsymbol{x}_{\mathcal{A}}$ at positions $\mathcal{A}$ we can express for a finite set of other positions $\mathcal{V} \backslash \mathcal{A}$ the conditional distribution of those values. This means that we are able, for each point $y \in \mathcal{V} \backslash \mathcal{A}$, to predict the mean and the variance of $\boldsymbol{x}_y$. Using conditional distribution for the Multivariate Gaussian Distribution (Deisenroth et al., 2018, p. 193), we are able to express the following :

$$P(\mathcal{X}_y | \boldsymbol{x}_{\mathcal{A}}) = \mathcal{N}(\mu_{y|\mathcal{A}}, K_{y|\mathcal{A}}) \tag{2.2}$$

$$\mu_{y|\mathcal{A}} = \mu_y + K_{y\mathcal{A}} K_{\mathcal{A}\mathcal{A}}^{-1} (y - \mu_y) \tag{2.3}$$

$$K_{y|\mathcal{A}} = K_{yy} - K_{y\mathcal{A}} K_{\mathcal{A}\mathcal{A}}^{-1} K_{\mathcal{A}y} \tag{2.4}$$

An important point to notice is that the predicted covariance for the point y is not dependent of the values measured at $\mathcal{A}$, this is really useful because it allows us to define the uncertainty at y without using only the covariance.

2.2.2 Scalable Gaussian Processes

The biggest weakness of Standard GPs is their complexity. For p training points, the algorithm requires the inversion of a $p \times p$ covariance matrix K_{pp}. Liu et al. (2018) gives an extensive review of all methods used to make the GPs more scalable. Those methods are evaluated with regards to their *scalability* and their *capability*.

We redefine notations used here to expose scalable Gaussian Processes. We consider n training points $\mathbf{X} = \{\boldsymbol{x}_i \in \mathbb{R}^d\}_{i=1}^n$, and observations $\boldsymbol{y} = \{y_i = y(\boldsymbol{x}_i) \in \mathbb{R}\}_{i=1}^n$. GP aims at inferring the function $f : \mathbb{R}^d \mapsto \mathbb{R}$ that describes the best the training data. This function belongs to the function space $\mathcal{GP}(\mathcal{M}(\boldsymbol{x}), \mathcal{K}(\boldsymbol{x}, \boldsymbol{x}'))$. We then replace the conditionals . In order to obtain the observations we formally add gaussian noise $\mathcal{N}(0, \epsilon_n I_n)$ to the inferred function values (we didn't need that in the previous section).

2.2.2.1 Global Approaches

The aim of global approximation is to reduce the size of the covariance matrix, in order to make the computation scalable. There are several ways to proceed: **subset of data** method where we reduce the size of the training set; the **sparse kernel** methods where we exploit a low-rank representation of the GP (**sparse approximation**).

The **subset of data method** (SoD) is a very simple strategy that only requires to create a subset of the original training data. The resulting cost of the GP is only of $\mathcal{O}(m^3)$ with m the number of points in the subset and $m << p$.

The **sparse kernel** approach is the idea to reduce the importance of the points far from each other and imposing sparsity of the covariance by setting to zero the elements of the matrix K_{pp} that are below a certain threshold. The resulting complexity being of $\mathcal{O}(\alpha p^3)$ with $0 < \alpha < 1$. The difficulty here is to ensure the positive definiteness of the resulting covariance. Methods for obtaining such results are discussed in the next section.

The **sparse approximation** methods are very diverse and are allowed by different kinds of approximations : the **approximation of the prior** ; the **approximation of the posterior** ; **structured sparse approximations**.

Prior approximation modifies the joint prior $p(f, f_*)$ between the training data f_* and the test data f by assuming independence between them knowing **inducing variables** f_m : $f_* \perp f \mid f_m$. We also define the Nyström notation for the covariance : $\boldsymbol{Q}_{ab} = \boldsymbol{K}_{am}\boldsymbol{K}_{mm}^{-1}\boldsymbol{K}_{mb}$. We are then able to express the conditional probabilities for $p(f|f_m) = \mathcal{N}(f \mid \boldsymbol{K}_{nm}\boldsymbol{K}_{mm}^{-1}f_m \, , \boldsymbol{K}_{nn} - \boldsymbol{Q}_{nn})$ and $p(f_*|f_m) = \mathcal{N}(f \mid \boldsymbol{K}_{*m}\boldsymbol{K}_{mm}^{-1}f_* \, , \boldsymbol{K}_{**} - \boldsymbol{Q}_{**})$ and approximate them to : $q(f|f_m)$ and $q(f_*|f_m)$ by replacing respectively the covariances by $\tilde{\boldsymbol{Q}}_{nn}$ and $\tilde{\boldsymbol{Q}}_{**}$. This approximation allows for a computation with a reduced complexity of $\mathcal{O}(mn^2)$.

In order to select an appropriate approximate covariance $\tilde{\boldsymbol{Q}}_{nn}$ and $\tilde{\boldsymbol{Q}}_{**}$, different methods are referenced such as SoR, DTC, FITC and PITC.
Subset of Regressor (SoR) fixes both covariances to 0, forcing a deterministic view.
In the *Deterministic Training Conditional* (DTC), imposes a deterministic training $\tilde{\boldsymbol{Q}}_{nn} = 0$ but keeps the exact test conditional $p(f_*|f_m)$.
Another approach is the *Fully Independent Training Conditional* (FITC). Here we keep exact the test conditional but we approximate the training by considering that each realisation is independent from each other : the $\{f_i\}_{i=1}^n$ are fully independent, and we take the covariance to be $\tilde{\boldsymbol{Q}}_{nn} = diag(\boldsymbol{K}_{nn} - \boldsymbol{Q}_{nn})$. This leads to better results than the two previous methods. In order to further improve this method the idea of *Partially Independent Training Conditional* (PITC) was introduced. Here we consider that not all the training points are independent, but only some of them. This leads to an independence by blocks, and an block-diagonal covariance approximation : $\tilde{\boldsymbol{Q}}_{nn} = blkdiag(\boldsymbol{K}_{nn} - \boldsymbol{Q}_{nn})$.
In those methods, the choice of the m *inducing points* is crucial to enable a good approximation.

Another way to see the problem is to **approximate the posterior** $p(f, f_*|\boldsymbol{y})$ of the GP instead of the prior $p(f, f_*)$, by replacing it with a variational distribution $q(f, f_*|\boldsymbol{y})$. The main method is called the *Variational Free Energy* (VFE), and has been developed by Titsias (2009). We optimise the variational parameters and hyperparameters of the approximation by minimising the KL divergence between the two distributions :

$$KL(q(f, f_*|\boldsymbol{y}) \mid\mid p(f, f_*|\boldsymbol{y})) = \log p(\boldsymbol{y}) - \left\langle \log \frac{p(f, f_*, \boldsymbol{y})}{q(f, f_*|\boldsymbol{y})} \right\rangle_{q(f, f_*|\boldsymbol{y})} \tag{2.5}$$

$$= \log p(\boldsymbol{y}) - F_q \tag{2.6}$$

As the KL divergence is positive, minimising this quantity is equivalent to maximising the Variational Free Energy (VFE): F_q (also called Evidence Lower Bound - ELBO). A tighter bound can be derived to replace F_q :

$$F_{VFE} = \log q_{DTC}(\boldsymbol{y}) - \frac{1}{2\sigma_\epsilon^2} tr(\boldsymbol{K}_{nn} - \boldsymbol{Q}_{nn}) \tag{2.7}$$

In the DTC we maximised the likelihood $q_{DTC}(\boldsymbol{y})$ and here we add only an additional term. This has 3 functions: it acts as a regulariser against over-fitting; it helps finding a good inducing set; it always improves the ELBO with increasing m.

Finally, the last global approach is the *structured sparse approximations*. The main idea is to speed up the computation of the inversion of the covariance matrix and multiplication by a vector: $\boldsymbol{K}_{nn}^{-1}\boldsymbol{y}$. For that fast *matrix vector multiplication* (MVM) is used, it solves the linear system by using conjugate gradients. Several improvements have been proposed, especially when the covariance matrix $\boldsymbol{K}_{nn}$ has some algebraic structure, such as in the *Kroenecker Methods* and the *Toeplitz Methods*. Those methods require also inputs having a grid structure, therefore not directly applicable in our project. In order to generalise those methods to any kind of data structure, the method of structured kernel interpolation (SKI) was also proposed by Wilson and Nickisch (2015). Inducing points are created using local linear interpolation techniques and constrained to the grid structure. This method has drawbacks such as exponential growth of inducing points in high dimension; discontinuous predictions and overconfident variance.

2.2.2.2 Local Approaches

Applying the principle de Divide and Conquer (D&C), the main idea is to use *local experts* to create a full GP representation. It has the main advantage to enable naturally *parallelisation* and to capture *non-stationary* features. We can use naive local experts (NLE), or more complex combinations of localised experts such as mixture of experts (MoE) or Products of Experts (PoE).

In naive local experts (NLE), each expert is trained on a different subset of the space. For a training set of points $\mathcal{D}$ and its subsets $\mathcal{D}_i$, we have the prediction of the expert $\mathcal{M}_i$ inside a subregion Ω_i : $\boldsymbol{x}_* \in \Omega_i$ such as $p(y_*|\, \mathcal{D}, \boldsymbol{x}_*) \approx p_i(y_*|\, \mathcal{D}_i, \boldsymbol{x}_*)$. In *inductive* NLE, the space is first partitioned and each expert is trained before choosing the appropriate one of the prediction. The partition is therefore fixed beforehand. In *transductive* NLE, once the prediction point $\boldsymbol{x}_*$ has been chosen, the expert $\mathcal{M}_*$ is trained locally on a subset $\mathcal{D}_*$ close to $\boldsymbol{x}_*$. A few drawbacks are the discontinuities at the boundaries between the subregions and the fact that it generalises badly to all the regions.

Note : This section will be developed further as the project explores those methods in practice especially MoE and PoE.

2.2.3 Covariance Matrix Estimation

We have seen how GPs are defined and made more scalable. In order to have good results, we need to have a good estimate of the covariance matrix between the points of our space.

In our specific case, we already have at our disposal a very dense network of measurement. With more than $100'000$ different locations we don't need to explore the space outside of those points. Unfortunately, the sample covariance is a very bad estimator of the true covariance in high dimensional settings such as ours (Pourahmadi, 2011).

We will see the properties that our covariance to accurately model our space, before discussing the best ways of estimating the covariance matrix estimation.

2.2.3.1 Properties of Covariance

By definition a covariance matrix must be **positive-definite** and **symmetrical** (Rasmussen and Williams, 2006, p. 80).

A covariance function between two inputs x and x' is **stationary** when it is invariant to translation. Thus, when it is a function of $x' - x$.
In a covariance function that is **isotropic**, we remove the dependence of the direction and, it becomes a function of the distance between the two points: $|x' - x|$. The isotropy of the covariance function is a stronger assumption than

In our problem, we can't assume that the process is stationary nor isotropic. Our space is 3-dimensional and not homogeneous. The presence of the buildings and other obstacles that likely to make environment variables less smooth (Paciorek and Schervish, 2004). Also, it has been shown by Krause et al. (2008) that non-stationary covariance matrixes give better results than stationary or isotropic. This makes us choose a non-stationary covariance matrix for the GPs of our problem.

2.2.3.2 Covariance Estimation

We consider the process Y in p different locations at T sampling times. $\boldsymbol{Y}_t = (Y_{1t}, \ldots, Y_{pt})$, with $t = 1 \ldots T$. We want to estimate the covariance matrix Σ.

Sample Covariance

The simplest estimator of the covariance matrix is the **sample covariance matrix $\boldsymbol{S}$** directly computed from the captured data.

$$\boldsymbol{S} = \frac{1}{T-1} \sum_{t=1}^{T} (\boldsymbol{Y}_t - \bar{\boldsymbol{Y}}) \cdot (\boldsymbol{Y}_t - \bar{\boldsymbol{Y}}_t)^T, \quad \bar{\boldsymbol{Y}} = \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{Y}_t \tag{2.8}$$

Unfortunately, this covariance is singular when $p >> T$ (Fan et al., 2015), and the accumulation of errors due to the number of parameters to estimate. One additional assumption that we need to make for this covariance is its the sparsity of the covariance which is needed to reduce the number of parameters and is often realistic in practice.

Estimating a Spatial Covariance

Nott and Dunsmuir (2002) proposed to fit local variograms in order to approaximate the covariance matrix. It is a method that was developped in Cressie (1991).

Another method is the use of spatial deformation of the kernel in order to obtain non-stationarity (Sampson and Guttorp, 1992).

Several different approaches are developed in the literature, notably the works of (Fan et al., 2015) and Guttorp and Sampson (1994) which is more focused on spatial data.
Note : This section needs further developments, in order to find an implement the best possible approach in *coordination* with the scalable Gaussian Processes.

2.3 Sensor Position Optimisation

Now that we have modelled the relationship between the positions using GPS we can establish an algorithm that was developed by Krause et al. (2008). The process of placing sensors in an optimal way is called in spatial statistics, *sampling* or *experimental design*. We want to find the optimal way to place a number of k sensors (indexed by $\mathcal{A}$) inside the set of possible sensor locations $\mathcal{S}$. So that we have $\mathcal{A} \subseteq \mathcal{S} \subseteq \mathcal{V}$.

For the rest of this section, we assume that we have at our disposal a good estimate of the covariance matrix between each point. In practice this is not that obvious as we have seen in section 2.2.3. The following is valid for any covariance matrix that is symmetric and positive-definite.

First, we will define how to characterise a good design in term of sensor placement. Then we will define the main optimisation algorithm and its improvements.

2.3.1 Placement Criterion

2.3.1.1 Entropy Criterion

Intuitively a good way of measuring uncertainty is the *entropy*. By observing the conditional entropy of the location where no sensor was placed $\mathcal{V}\backslash\mathcal{A}$, we can estimate the uncertainty remaining for those locations. We define the following conditional entropy of the un-instrumented location knowing the instrumented ones (Cover and Thomas, 1991, p. 16) :

$$H(\mathcal{X}_{\mathcal{V}\backslash\mathcal{A}}|\ \mathcal{X}_{\mathcal{A}}) = \mathbb{E}_{p(\boldsymbol{x}_{\mathcal{V}\backslash\mathcal{A}},\boldsymbol{x}_{\mathcal{A}})} \log p(\mathcal{X}_{\mathcal{V}\backslash\mathcal{A}}|\ \mathcal{X}_{\mathcal{A}}) \tag{2.9}$$

$$H(\mathcal{X}_{\mathcal{V}\backslash\mathcal{A}}|\ \mathcal{X}_{\mathcal{A}}) = -\sum_{\boldsymbol{x}_{\mathcal{V}\backslash\mathcal{A}}\in\mathcal{X}_{\mathcal{V}\backslash\mathcal{A}}} \sum_{\boldsymbol{x}_{\mathcal{A}}\in\mathcal{X}_{\mathcal{A}}} p(\boldsymbol{x}_{\mathcal{V}\backslash\mathcal{A}}, \boldsymbol{x}_{\mathcal{A}}) \log p(\boldsymbol{x}_{\mathcal{V}\backslash\mathcal{A}}|\boldsymbol{x}_{\mathcal{A}}) \tag{2.10}$$

$$H(\mathcal{X}_{\mathcal{V}\backslash\mathcal{A}}|\ \mathcal{X}_{\mathcal{A}}) = -\int p(\boldsymbol{x}_{\mathcal{V}\backslash\mathcal{A}}, \boldsymbol{x}_{\mathcal{A}}) \log p(\boldsymbol{x}_{\mathcal{V}\backslash\mathcal{A}}|\boldsymbol{x}_{\mathcal{A}})\, d\boldsymbol{x}_{\mathcal{V}\backslash\mathcal{A}}\, d\boldsymbol{x}_{\mathcal{A}} \tag{2.11}$$

For the specific case of the *Multivariate Gaussian Distribution*, Krause et al. (2008) gives us the expression of the entropy of a point $y \in \mathcal{V}\backslash\mathcal{A}$ conditioned by the set $\mathcal{A}$ in a closed form depending exclusively on the covariance between those elements: $K_{y|\mathcal{A}}$. Thus we have :

$$H(\mathcal{X}_y|\ \mathcal{X}_{\mathcal{A}}) = \frac{1}{2}\log K_{y|\mathcal{A}} + \frac{1}{2}(1 + \log 2\pi) \tag{2.12}$$

This formulation is extremely useful because we can directly use the expression of the covariance given by the *Gaussian Process* expressed previously (2.4).

This conditional entropy can also be expressed using the *chain rule* (Cover and Thomas, 1991, p. 16) :

$$\begin{aligned} H(\mathcal{X}_{\mathcal{V}\backslash\mathcal{A}}|\ \mathcal{X}_{\mathcal{A}}) &= H(\mathcal{X}_{\mathcal{V}\backslash\mathcal{A}},\ \mathcal{X}_{\mathcal{A}}) - H(\mathcal{X}_{\mathcal{A}}) && (2.13)\\ &= H(\mathcal{X}_{\mathcal{V}}) - H(\mathcal{X}_{\mathcal{A}}) && (2.14) \end{aligned}$$

The optimal set of sensors $\mathcal{A}^*$ with size $|\mathcal{A}^*| = k$, is then defined for the minimum of this entropy. If we minimise this quantity we will reduce the uncertainty on the un-instrumented locations $\mathcal{V}\backslash\mathcal{A}$:

$$\begin{aligned} \mathcal{A}^* &= \arg\min\nolimits_{\mathcal{A}\subseteq\mathcal{V}:|\mathcal{A}|=k} H(\mathcal{X}_{\mathcal{V}\backslash\mathcal{A}}|\ \mathcal{X}_{\mathcal{A}}) && (2.15)\\ &= \arg\min\nolimits_{\mathcal{A}\subseteq\mathcal{V}:|\mathcal{A}|=k} H(\mathcal{X}_{\mathcal{V}}) - H(\mathcal{X}_{\mathcal{A}}) && (2.16)\\ &= \arg\max\nolimits_{\mathcal{A}\subseteq\mathcal{V}:|\mathcal{A}|=k} H(\mathcal{X}_{\mathcal{A}}) && (2.17) \end{aligned}$$

2.3.1.2 Mutual Information Criterion

The Entropy criterion provides an intuitive way to solve the problem, unfortunately, during experiments referenced by Krause et al. (2008), it was noted that this criterion has a tendency to induce placement at the border of the space, and therefore wastes a lot of precious information. This is due to the fact that the entropy criterion is *indirect* because it measures the uncertainty of the selected sensor position, instead of measuring the uncertainty of every other location of the space (which are the ones we are interested in).

An other criterion was proposed by Caselton and Zidek (1984) : the *Mutual Information* (MI) Criterion. We try to maximise the mutual information between the set of selected sensors $\mathcal{A}$ and the rest of the space $\mathcal{V}\backslash\mathcal{A}$. Using the definitions of MI provided by Cover and Thomas (1991, p. 19) :

$$\mathcal{A}^* = \arg\max_{\mathcal{A}\subseteq\mathcal{V}:|\mathcal{A}|=k} I(\mathcal{X}_{\mathcal{V}\backslash\mathcal{A}},\ \mathcal{X}_{\mathcal{A}}) \tag{2.18}$$

$$= \arg\max_{\mathcal{A}\subseteq\mathcal{V}:|\mathcal{A}|=k} H(\mathcal{X}_{\mathcal{V}\backslash\mathcal{A}}) - H(\mathcal{X}_{\mathcal{V}\backslash\mathcal{A}}|\mathcal{X}_{\mathcal{A}}) \tag{2.19}$$

Experimentally, Krause et al. (2008) explain that the mutual information outperforms entropy placement optimisation. They also argue that this criterion relies heavily on the quality of the model $P(\mathcal{X}_{\mathcal{V}})$ (i.e. how the covariance is modelled, see section 2.2.3) for giving good results.

2.3.2 Approximation Algorithm

As stated by Krause et al. (2008), the problem is a *NP-complete problem*. Therefore we present here an algorithm that is able to approximate in polynomial time the optimal solution, with a constant factor guarantee.

Let us define the initial sensor set $\mathcal{A}_0 = \emptyset$. At each iteration we have a new sensor set : $\mathcal{A}$, and for a point y, we define : $\bar{\mathcal{A}} = \mathcal{V}\backslash(\mathcal{A}\cup y)$. We also have the set of positions that can be selected as sensors : $\mathcal{S}$, and the associated set : $\mathcal{U} = \mathcal{V}\backslash\mathcal{S}$.

The idea is to greedily add sensors until we reach the wanted number (k). This greedy approach is enabled by the use of the *chain rule* of entropy. Starting from $\mathcal{A} = \mathcal{A}_0$, at each iteration we add to $\mathcal{A}$ the point $y \in \mathcal{S}\backslash A$ which decreases the least the current MI :

$$y* = \arg\max_{y\in\mathcal{S}\backslash A} MI(\mathcal{A}\cup y, \bar{\mathcal{A}}) - MI(\mathcal{A}, \mathcal{V}\backslash\mathcal{A}) \tag{2.20}$$

$$= \arg\max_{y\in\mathcal{S}\backslash A} H(y|\mathcal{A}) - H(y|\bar{\mathcal{A}}) \tag{2.21}$$

In our specific case with the Multivariate Gaussian Distribution, we can take the formulation presented in equation 2.12 and rewrite the objective to :

$$y* = \arg\max_{y\in\mathcal{S}\backslash A} \frac{1}{2}\log K_{y|\mathcal{A}} - \frac{1}{2}\log K_{y|\bar{\mathcal{A}}} \tag{2.22}$$

$$= \arg\max_{y\in\mathcal{S}\backslash A} \log\left(\frac{K_{y|\mathcal{A}}}{K_{y|\bar{\mathcal{A}}}}\right) \tag{2.23}$$

$$= \arg\max_{y\in\mathcal{S}\backslash A} \frac{K_{y|\mathcal{A}}}{K_{y|\bar{\mathcal{A}}}} \tag{2.24}$$

Here we can use the GP that we have defined earlier in equation 2.4 to finally write the problem to solve at each iteration of the algorithm :

$$y* = \arg\max_{y\in\mathcal{S}\backslash A} \frac{K_{yy} - K_{y\mathcal{A}}K_{\mathcal{A}\mathcal{A}}^{-1}K_{\mathcal{A}y}}{K_{yy} - K_{y\bar{\mathcal{A}}}K_{\bar{\mathcal{A}}\bar{\mathcal{A}}}^{-1}K_{\bar{\mathcal{A}}y}} \tag{2.25}$$

Then we update the set of sensors such that $\mathcal{A} = \mathcal{A} \cup y^*$, and then restart the process until $|\mathcal{A}| = k$. It is described in algorithm 1

```
Data: Covariance matrix $K_{\mathcal{V}\mathcal{V}}$ , $k$, $\mathcal{V}$
Result: Sensor Selection $\mathcal{A}$
begin;
for $j \leftarrow 1$ to $k$ do
    for $y \in \mathcal{S}\backslash\mathcal{A}$ do
        $\delta_y \leftarrow \frac{K_{yy} - K_{y\mathcal{A}} K^{-1}_{\mathcal{A}\mathcal{A}} K_{\mathcal{A}y}}{K_{yy} - K_{y\bar{\mathcal{A}}} K^{-1}_{\bar{\mathcal{A}}\bar{\mathcal{A}}} K_{\bar{\mathcal{A}}y}}$
    end
    $y^* \leftarrow \arg\max_{y \in \mathcal{S}\backslash\mathcal{A}} \delta_y$
    $\mathcal{A} \leftarrow \mathcal{A} \cup y^*$
end
```

Algorithm 1: Greedy Algorithm

This algorithm computes a solution that is very close to the optimal one if the discretisation of the space is small enough. The bound of the solution obtained is approximately 63% of the optimal solution. If the true optimal set is $\mathcal{A}^*$ and $\hat{\mathcal{A}}$ is the solution returned by the greedy algorithm, then Krause et al. (2008) proves, for a small $\epsilon > 0$, that :

$$MI(\hat{\mathcal{A}}) \geq (1 - \frac{1}{e}) \cdot MI(\mathcal{A}^*) - k\epsilon \tag{2.26}$$

To prove that they use the notion of *submodularity* (Nemhauser et al., 1978) applied to the $MI(\cdot)$ function. Intuitively this represents the notion of *diminishing returns* : adding a sensor to a small set of sensors has more benefits than adding a sensor to a large set of sensors.

2.3.3 Improvements over the Algorithm

We have exposed the main greedy algorithm to solve that optimisation problem. Krause et al. (2008) explains that complexity is a big issue for scaling this algorithm up. If we consider that the number of locations in our space is $|\mathcal{V}| = n$, and that the number of sensors to place is k, the complexity of the main algorithm is $\mathcal{O}(kn^4)$. They propose two solutions to this issue: a *lazy procedure* that cuts the complexity to $\mathcal{O}(kn^3)$ and a *local kernel* strategy that reduces it to $\mathcal{O}(kn)$

2.3.3.1 Lazy Procedure

This procedure uses strategically the notion of *submodularity* and *priority queues*. We can describe it intuitively: When a sensor is selected y^*, the other nearby points will have de decreased δ_y and will, therefore, be less desirable. Therefore if we maintain a priority queue with the ordered values of δ_y at each step we will take the top values and update them to the current value successively. If the current value needs to be updated and the location is close to the previous optimum, it would have a small

δ_y and be send back in the queue. If we meet a point which has been updated and is still a the top of the queue, it means that this point is our new optimum. This technique can be efficiently applied using **binary heaps**. This algorithm is described in the algorithm 2.

```
Data: Covariance matrix $K_{\mathcal{V}\mathcal{V}}$ , $k$, $\mathcal{V}$
Result: Sensor Selection $\mathcal{A}$
initialisation;
$\mathcal{A} = \emptyset$
foreach $y \in \mathcal{S}$ do $\delta_y \leftarrow +\infty$;
begin;
for $j \leftarrow 1$ to $k$ do
    foreach $y \in \mathcal{S}\backslash\mathcal{A}$ do $current_y \leftarrow$ False ;
    while True do
        $y^* \leftarrow \arg\max_{y\in\mathcal{S}\backslash\mathcal{A}}\delta_y$
        if $current_{y^*}$ then break;
        $\delta_{y^*}$ is updated with 2.25
        $current_y \leftarrow$ True
    end
    $\mathcal{A} \leftarrow \mathcal{A} \cup y^*$
end
```

Algorithm 2: Lazy Algorithm

2.3.3.2 Local Kernels

This procedure takes advantage of the structure of the covariance matrix. For many GPs, correlation decreases exponentially with the distance between points. So, the idea here is to truncate the covariance matrix to points that are the most correlated.

If we want to compute the covariance between a point of interest y and a set of points $\mathcal{B} \subset \mathcal{V}$ we have the original covariance matrix $K_{y\mathcal{B}}$. When we remove from the set $\mathcal{B}$, the set of elements $x \in \mathcal{S}$ such that $|\mathcal{K}(y,x)| > \epsilon$ we define the new set $\tilde{\mathcal{B}}_y = N(y,\epsilon)$. This set is defined such as it contains less than d locations : $|N(y,\epsilon)| \leq d$. The truncated covariance associated with this set is named : $\tilde{K}_{y\mathcal{B}}$. Finally, we define the approximate conditional entropy $\tilde{H}_\epsilon(y|\mathcal{X}) \simeq H(y|\mathcal{X})$, computed with truncated covariance of the points of $N(y,\epsilon)$. The δ_y values are initialised by taking the difference between the true entropy and the truncated covariance, as :

$$\delta_y = \tilde{H}_\epsilon(y|\mathcal{A}) - \tilde{H}_\epsilon(y|\bar{\mathcal{A}}) \tag{2.27}$$

$$= H(y) - \tilde{H}_\epsilon(y|\mathcal{V}\backslash y) \tag{2.28}$$

Or equivalently by keeping the previous notations:

$$\delta_y = \frac{K_{yy}}{K_{yy} - \tilde{K}_{y\mathcal{V}\backslash y}\tilde{K}^{-1}_{\mathcal{V}\backslash y\mathcal{V}\backslash y}\tilde{K}_{\mathcal{V}\backslash yy}} \tag{2.29}$$

As for the other iterations (when $\mathcal{A} \neq \emptyset$), we have :

$$\delta_y = \tilde{H}_\epsilon(y|\mathcal{A}) - \tilde{H}_\epsilon(y|\bar{\mathcal{A}}) \tag{2.30}$$

$$= \frac{K_{yy} - \tilde{K}_{y\mathcal{A}}\tilde{K}^{-1}_{\mathcal{A}\mathcal{A}}\tilde{K}_{\mathcal{A}y}}{\tilde{K}_{yy} - \tilde{K}_{y\bar{\mathcal{A}}}\tilde{K}^{-1}_{\bar{\mathcal{A}}\bar{\mathcal{A}}}\tilde{K}_{\bar{\mathcal{A}}y}} \tag{2.31}$$

We explicitly define the steps in algorithm 3.

```
Data: Covariance matrix $K_{\mathcal{V}\mathcal{V}}$ , $k$, $\mathcal{V}$, $\epsilon$
Result: Sensor Selection $\mathcal{A}$
initialisation;
$\mathcal{A} = \emptyset$
foreach $y \in \mathcal{S}$ do $\delta_y \leftarrow$ 2.29 ;
begin;
for $j \leftarrow 1$ to $k$ do
    foreach $y \in \mathcal{S} \backslash \mathcal{A}$ do $current_y \leftarrow$ False ;
    while True do
        $y^* \leftarrow \arg\max_{y \in \mathcal{S} \backslash A} \delta_y$
        $\mathcal{A} \leftarrow \mathcal{A} \cup y^*$
        foreach $y \in N(y^*, \epsilon)$ do $\delta_y \leftarrow$ 2.31 ;
    end
end
```

Algorithm 3: Local Kernel Algorithm

Krause et al. (2008) proves that this algorithm approximates the optimal solution with complexity $\mathcal{O}(nd^3 + nk + kd^4)$. The solution found by this algorithm is close to the real optimum within given bounds.

This two strategies can be combined in order to make the problem even more scalable.

Chapter 3

Progress

This chapter exposes the work done until now in the project. It will focus on the choices I have made, the stages of developpment and the questions remaining. All the codes are available on the Github repository of the project.

3.1 Data Exploration

As the main dataset, we use the simulation results on an unstructured mesh of measurements with a very high density in the low middle part of the space. The simulation used has $100'040$ points over 988 timestamps. For the purpose of testing our codes and algorithms, we are going to use subset of the whole dataset containing a few hundreds to a few thousands of points. I have made the choice to first use a subset of the whole dataset containing $2'488$ point and created by cropping the space to a cube of $30 \times 30 \times 30 \times$ at the center of the original space.
Several fields are present in the simulation : the **pressure**, the **tracer** concentration, the **background tracer** concentration and the **velocity**. We represent in the next figures, two cuts made to visualise the tracer and the tracer background.

The **tracer** represents the propagation of a pollutant generated from the centre of the domain at ground level. It aims a representing a busy intersection.

The **background tracer** represents pollution waves common in urban environment. Its characteristic generation equation is periodic and given by :

$$C(t) = \frac{1}{2}\left(\sin\left(\frac{2\pi t}{T}\right) + 1\right) \tag{3.1}$$

As we can see by observing the time propagation of those fields, the wind is pushing the pollutants in a certain direction. Because of this, be observe that the **tracer** concentration is mainly visible downwind, in contrast of the **background tracer** which has visible propagation over the whole space. We can therefore already deduce that the **background tracer** contains much more information and should be used for optimising the sensor positions.

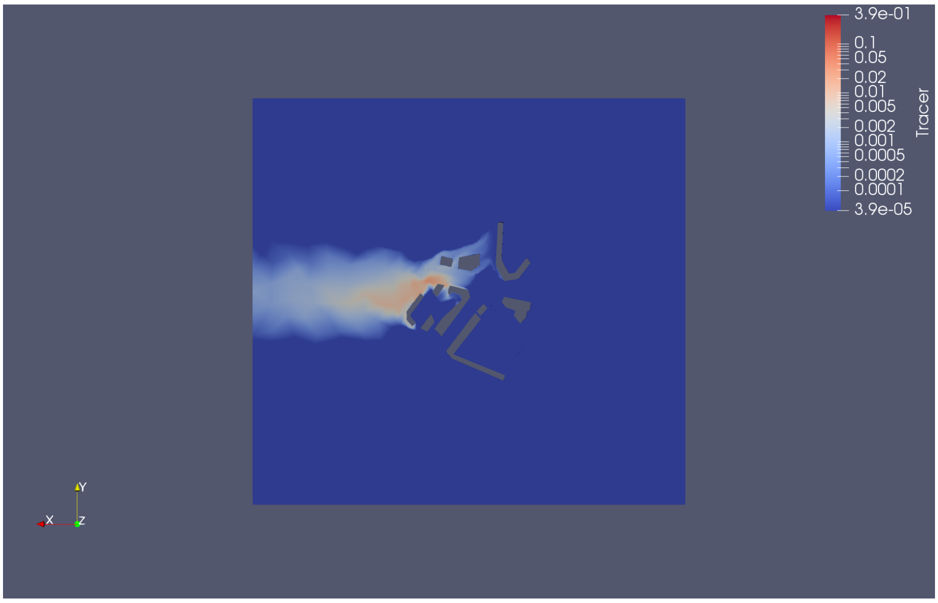

Figure 3.1: Cut of tracer at t = 639

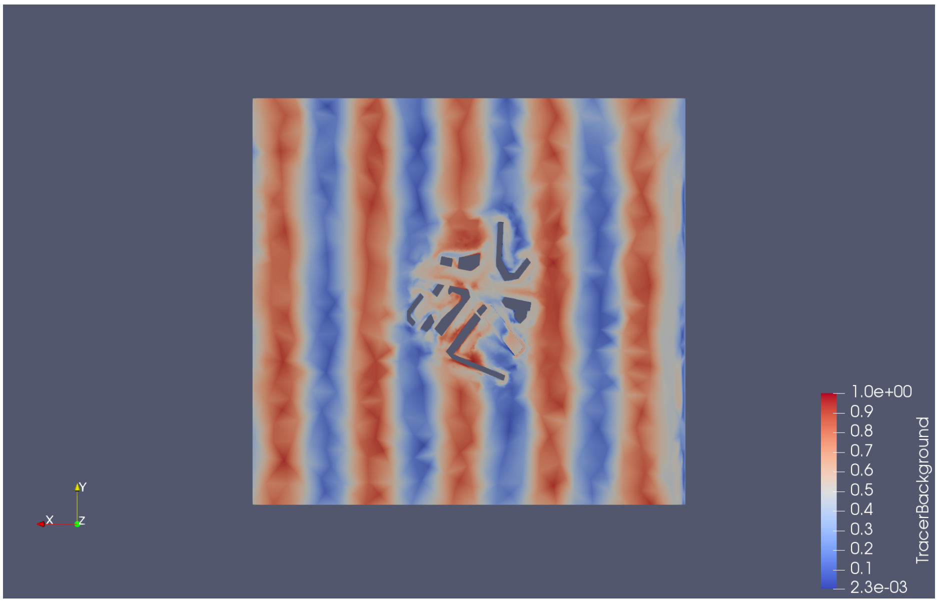

Figure 3.2: Cut of tracer background at t = 106

3.2 Implementation

In this section I will quickly review the functionalities that I have implemented and that are working to this day. I relied on Python 3.7, various libraries that I ran my code on the cluster of the DSI.

3.2.1 Initialisation

The raw data is contained in VTK files. I have implemented functions in python that allows the importation, the cropping of the space, the extraction of the fields and interest and the location of the points and the saving in files for making quicker the loading procedure.

3.2.2 Modeling

I have implemented the full GP training and estimation directly using numpy. It this implementation I have chosen to use a kernel with arbitrary parameters in order to test the accuracy of the optimisation.

After this I have used the GPy library (GPy, 2014) for its full GP implementation and its hyper-parameters optimisation functionalities. The kernel parameters here were optimised using the integrated methods and the model was trained on the full subset of data.

I have also implemented using GPy library a spare GP modelisation. GPy includes an optimised implementation of the **VFE** sparse GP. It enables the optimisation of the variational parameters and the hyper-parameters such as the position of the inducing points.

3.2.3 Optimisation

I have used the previously trained GPs models in the algorithms described earlier. I have implemented in python two of the proposed optimisation algorithms. The naive greedy approach (algorithm 1).
The Lazy approach developed using python heaps. (algorithm 2). This version is as expected much faster than the standard version.

For illustration we plot in the following figure 3.3 the results obtained with full GP of a small subset of the space described earlier for a number of sensors $k = 10$. On this subspace, I have made the choice to compute an isotropic covariance matrix (Exponential kernel, Matern 3/2 and 5/2) with a length-scale of $l = 3$ for testing the performance of the optimisation algorithm. The sensors optimal positions are marked in red.

3.2.4 Display and Exportation

In order to visualise the results of the optimisation, I have implemented function that saves the optimal locations in VTK files for an easier visualisation. I have also implemented function enabling visualisation with matplotlib of sections of the space.

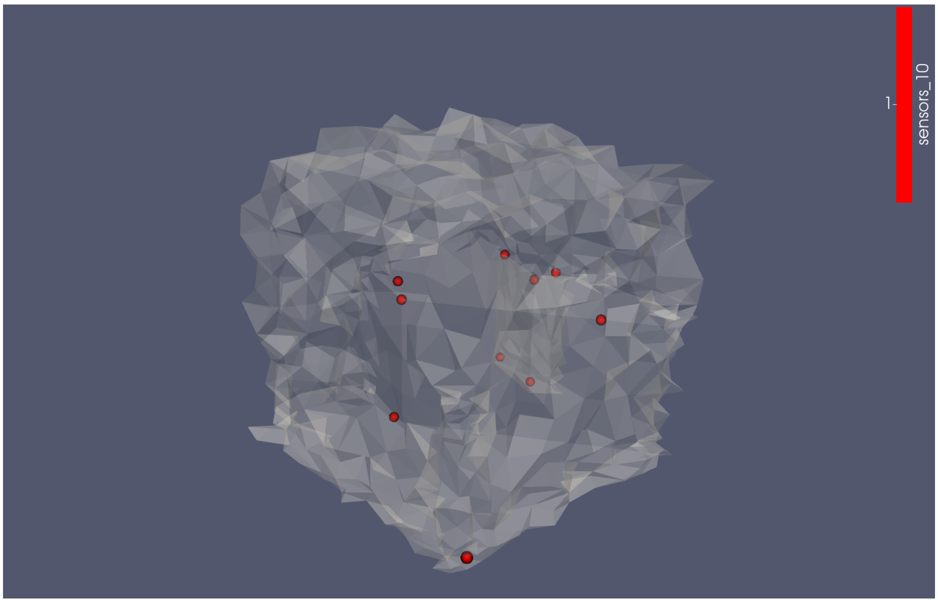

Figure 3.3: Position of the optimal set of 10 sensors

3.3 Discussion

In this section, I will try to show and justify the approach I have taken, as well as the remaining questions and problems to be solved.

3.3.1 Summary of the GP Sensor Optimisation

The methods developed by Krause et al. (2008) are using the full Gaussian process in to greedily place sensors. The algorithm consists in the estimation of the best "new" sensor to add the set of existing sensor based on a mutual information gain criterion. This criterion can be computed with two Gaussian processes. One that estimates the covariance of the sensor candidate y given the set of previously selected sensors $\mathcal{A}$: $K_{y|\mathcal{A}}$. The other GP allows the computation of the covariance of the candidate y given the rest of the locations in the space $\bar{\mathcal{A}} = \mathcal{V}\backslash(\mathcal{A} \cup y)$: $K_{y|\bar{\mathcal{A}}}$. We recall the problem that is to be solved is :

$$y^* = \arg\max{}_{y \in \mathcal{S} \backslash A} \frac{K_{y|\mathcal{A}}}{K_{y|\bar{\mathcal{A}}}} \tag{3.2}$$

This has to be computed theoretically for every candidate point of the dataset, and this for every sensor we add to A. The naive version is described in 1 and a lazy version of the algorithm is available at algorithm 2.

3.3.2 Covariance Estimation

The GP estimation of the conditional covariances mentioned previously relies heavily on the initial joint covariance.

$$K_{y|\mathcal{A}} = K_{yy} - K_{y\mathcal{A}} K_{\mathcal{A}\mathcal{A}}^{-1} K_{\mathcal{A}y} \tag{3.3}$$

This is why it is really important to have a proper way to estimate the covariance in our space of interest.
Those are the three main approaches I have identified initially.

1. Use the sample covariance of the given data
2. Use a non-stationary covariance that we learn with the data
3. Use isotropic and stationary covariance and learn its parameters with the GP optimisation.

3.3.2.1 Sample Covariance

First, the use of the sample covariance as an estimator of the true covariance has proven not to work. The resulting covariance is singular and therefore can't be inverted for the use in 3.3. Also, it is, in general, a poor estimator of the real covariance when the number of points is much larger than the number of samples. The memory needs to store such a full matrix is also not practical as on the cluster it used almost 75% of all the memory.

3.3.2.2 Non-Stationary Covariance

Secondly, the use of non-stationary covariance estimator as suggested by Krause et al. (2008) is probably the best solution for our problem.
Several approaches have been developed in the literature about this specific subject. Many methods were design in the Geostatistical field where the main application is the interpolation a small number of sensor points to the whole space (kriging).
There are also interesting studies on the estimation of sparse covariance matrixes.

Explain that I use the VFE only for predicting the canditate given the whole dataset. The other GP is tracktable because it is conditioned by the number of sensors already placed.

3.3.2.3 Isotropic Covariance

For now, I have decided to keep the kernel stationary and isotropic in order to focus on the scalability of the GPs and the algorithm. The main advantage of this kind of covariance is that is relies uniquely on parameters that can be optimised in the training of the GP, whereas data-driven covariance requires much more memory and computation.

3.3.3 Gaussian Process

Initially I implemented the full GP version proposed in the paper of Krause et al. (2008). For a small dataset it gives consistent results.

I have then explored ways of making GPs more scalable and focused on the VFE sparse Gaussian Process. For larger datasets I am able to optimise the hyperpameters and the positions of the induction points before using this model.

I use the sparse GP

Chapter 4

Conclusion

In this report, we have stated some of the research that was done in order to solve the sensor position optimisation problem.

The biggest challenge here is to make the algorithm scalable for optimising over $100'000$ sensor locations. It requires to develop strategies to make the GPs scalable. The main other challenge is to find a method to accurately estimate the covariance matrix.

I have implemented some of the optimisation algorithm proposed on a smaller dataset and with a poor estimation of the covariance function. It has given consistent results and has proven that once the main challenges, described earlier are solved, we will have a good optimisation algorithm for that kind of setups.

Bibliography

Arcucci, R., Pain, C., and Guo, Y.-K. (2018). Effective variational data assimilation in air-pollution prediction. *Big Data Mining and Analytics*, 1(4):297–307. pages 3

Caselton, W. and Zidek, J. (1984). Optimal monitoring network designs. *Statistics & Probability Letters*, 2(4):223–227. pages 10

Cover, T. M. and Thomas, J. A. (1991). *Elements of information theory*. Wiley series in telecommunications. Wiley, New York. pages 9, 10

Cressie, N. A. C. (1991). *Statistics for spatial data*. Wiley series in probability and mathematical statistics. Wiley, New York. pages 3, 8

Deisenroth, M. P., Faisal, A., and Ong, C. S. (2018). Mathematics for Machine Learning. page 43. pages 4

Fan, J., Liao, Y., and Liu, H. (2015). An Overview on the Estimation of Large Covariance and Precision Matrices. *arXiv:1504.02995 [stat]*. arXiv: 1504.02995. pages 8

GPy (2014). GPy: Gaussian process framework in python. pages 16

Guttorp, P. and Sampson, P. D. (1994). 20 Methods for estimating heterogeneous spatial covariance functions with environmental applications. In *Handbook of Statistics*, volume 12, pages 661–689. Elsevier. pages 8

Krause, A., Singh, A., and Guestrin, C. (2008). Near-Optimal Sensor Placements in Gaussian Processes: Theory, Efficient Algorithms and Empirical Studies. page 50. pages 7, 8, 9, 10, 11, 13, 17, 18, 19

Liu, H., Ong, Y.-S., Shen, X., and Cai, J. (2018). When Gaussian Process Meets Big Data: A Review of Scalable GPs. *arXiv:1807.01065 [cs, stat]*. arXiv: 1807.01065. pages 4

Nemhauser, G. L., Wolsey, L. A., and Fisher, M. L. (1978). An analysis of approximations for maximizing submodular set functions—I. *Mathematical Programming*, 14(1):265–294. pages 11

Nott, D. J. and Dunsmuir, W. T. M. (2002). Estimation of nonstationary spatial covariance structure. *Biometrika*, 89(4):819–829. pages 8

Paciorek, C. J. and Schervish, M. J. (2004). Nonstationary Covariance Functions for Gaussian Process Regression. In Thrun, S., Saul, L. K., and Schölkopf, B., editors, *Advances in Neural Information Processing Systems 16*, pages 273–280. MIT Press. pages 7

Pourahmadi, M. (2011). Covariance Estimation: The GLM and Regularization Perspectives. *Statistical Science*, 26(3):369–387. arXiv: 1202.1661. pages 7

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian processes for machine* *learning*. Adaptive computation and machine learning. MIT Press, Cambridge, Mass. OCLC: ocm61285753. pages 3, 4, 7

Sampson, P. D. and Guttorp, P. (1992). Nonparametric Estimation of Nonstationary Spatial Covariance Structure. *Journal of the American Statistical Association*, 87(417):108–119. pages 8

Song, J., Fan, S., Lin, W., Mottet, L., Woodward, H., Davies Wykes, M., Arcucci, R., Xiao, D., Debay, J.-E., ApSimon, H., Aristodemou, E., Birch, D., Carpentieri, M., Fang, F., Herzog, M., Hunt, G. R., Jones, R. L., Pain, C., Pavlidis, D., Robins, A. G., Short, C. A., and Linden, P. F. (2018). Natural ventilation in cities: the implications of fluid mechanics. *Building Research & Information*, 46(8):809–828. pages 2

Titsias, M. K. (2009). Variational Learning of Inducing Variables in Sparse Gaussian Processes. page 8. pages 5

Wilson, A. G. and Nickisch, H. (2015). Kernel Interpolation for Scalable Structured Gaussian Processes (KISS-GP). page 10. pages 6