Imperial College London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

# Gaussian Processes for Optimal Sensor Position

## MSc Project Report

*Author:*
Adrian LÖWENSTEIN

*Supervisor:*
Dr. Rossella ARCUCCI

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

September 5, 2019

# Abstract

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 apply the results by means of a data assimilation with the data at the optimised positions.

# Acknowledgment

I would like to thank *Imperial College London* for the opportunity I was given to pursue my studies this prestigious university.

I would like to thank my supervisor Dr. Rossella Arcucci of the *Data Science Institute*, for her help and wise counselling throughout the project.

An finally, I would like to thank my girlfriend, and friends from Imperial College, who supported me during this project.

# Contents

# Chapter 1

# Introduction

The optimisation of sensor position is a very common problem in many domains. The reliability of very complex systems often depends on the choice of sensor position. The cost induced by an incorrect or supernumerary sensor placement can be very important. There is, therefore, a real need for an optimal sensor placement methodology in the industry and research applications.

An approach that has been proven to give a near-optimal solution to this complicated issue, is the one developed by Krause et al. (2008). It relies on the mathematical concept of the Gaussian Process (GP), a very powerful machine-learning tool. Based on the assumption that the observation data can be described as a Multivariate Gaussian Distribution, we can predict for any point a value and a confidence interval. Since the '70s, Gaussian Processes have been applied successfully to the Geostatistical and Meteorology fields for prediction, and the idea of using them to place sensors was very novel.

For this project, we aim at applying this methodology to air pollution measurement and simulation, coming from the MAGIC (Managing Air for Green Inner Cities) project. We would like to optimise the position of a small number of sensors that will optimally represent the physical phenomenon. We will take the discrete set of locations on which the pollution has been simulated and will select optimally a small number of sensors in this set.

In this application, we will meet several challenges that need to be addressed. The main one is the scalability of the methodology. The algorithm has been proven to be near-optimal when the set of sensor candidates is of the order to $1'000$ points. In our case, we will work on a set $100$ times larger than this. The bottleneck of the algorithm is the computational cost of the Gaussian Processes, thus we are going to explore solutions improving its scalability. The other challenge is to represent accurately the data, and most importantly the covariance, as the GP relies on it for the prediction. We will explore some solutions that will make the optimisation problem computable in our application.

Once the optimal set of locations is found, we could use it for the purpose of Data

Assimilation (DA). This algorithm allows to enhance the precision of a simulation by taking into consideration real observations. We could observe the effect of the points chosen on a DA simulation.

The methods developed in the project are completely general, even if we are using them in the specific context of Tracers in MAGIC. They can be used for any type of data and for various configurations. The algorithms that are available are not directly usable in the context of big cities. We are therefore providing several contributions to make the algorithms usable. Our contributions are listed as follows:

- Implementation in Python 3.7 of several near-optimal algorithms for positioning sensors.
- Implementation of Shrinkage Algorithms for the computation of a Valid Covariance Matrix.
- Methodology for the pre-selection of locations based on field values and human reachability.
- Methodology based on the approximation of GPs using Truncated Singular Values Decomposition (TSVD).
- Experimentation with real case scenarios taken from the MAGIC research project.

In this project, we will first review all the background information and the literature necessary. We will then develop the details of our implementation, before detailing and analysing the results of our optimisation. Finally we will apply our results to the field of Data Assimilation.

# 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 can be applied in the **Managing Air for Green Inner Cities** (**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** matrices which need to be properly estimated. Finally, those tools will enable us to create the **optimisation** algorithms for a near-optimal sensor positioning.

## 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 those systems increases the heat and the pollution levels, inducing an increased need for the HVAC and leading to a vicious circle. The MAGIC project aims at breaking this 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, an 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 out-

door environment. Further works include the development of reduced-order modelling (ROM) to make the simulations faster 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 improvement of those methods could be the optimisation of the position of the sensors which provide information for the DA.

## 2.2 Gaussian Processes

In this section, 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, 1993). Applied in particular in Geostatistics with methods known as **kriging**, 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, especially in the bayesian optimisation of hyperparameters.

For our problem, we will be modelling the data of the sensor with GPs and the assumption that the data is normally distributed.

### 2.2.1 Multivariate Gaussian Distribution

The GP is the elementary tool in our project and our optimisation algorithm. In the space of the simulation, we consider that we have a number of sensors measuring important quantities, such as temperature, pressure, the speed of the wind or the concentration of a pollutant at a given position. We make the fundamental assumption that all the measured field have a distribution that is has a *multivariate Gaussian joint distribution*, and that the samples taken are independant. Our data must be independant and identically distributed (i.i.d). If we define the the associated random variable as $\mathcal{X}_\mathcal{V}$ 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:

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

### 2.2.2 Prediction with Gaussian Processes

Let us still consider that we have the set of locations $\mathcal{V}$ and a set of sensors $\mathcal{A} \subset \mathcal{V}$. In order to predict the quantity at positions were we have no sensors ($\mathcal{V} \backslash \mathcal{A}$) we can

use a Gaussian Process. It allows us to infer a function belonging 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 at of the state 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 can 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 state values measured at $\mathcal{A}$, this is very useful, as it allows us to define the prediction uncertainty at $y$ by knowing only the prior covariance.

### 2.2.3 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 explore 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).

#### Subset of Data

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 issue being able to select locations that represents accurately all the observation data.

### Sparse Kernel

The **sparse kernel** approach is based on the idea of reducing the importance of the points far from each other (not very correlated) 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. This idea is used in the third algorithm proposed by Krause et al. (2008), that we will review later in this chapter.

### Low-Rank Approximation

In Foster et al. (2009), the idea of using the Truncated Singular Values Decomposition (TSVD) as a method of obtaining a low-rank, well-conditioned version of the covariance matrix was developed. The inconvenient of this method is that the TSVD is an operation of complexity $\mathcal{O}(n^3)$. We will propose in the following chapter a solution inspired by this method.

### Sparse Approximations

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.

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 a block-diagonal covariance approximation: $\tilde{\boldsymbol{Q}}_{nn} = blkdiag(\boldsymbol{K}_{nn} - \boldsymbol{Q}_{nn})$.

In all 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 using this approach is called the *Variational Free Energy* (VFE), and has been developed by Titsias (2009). We optimise the variational parameters and hyper-parameters of the approximation by minimising the KL divergence between the two distributions:

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

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

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, another 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. 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.

**Remarks on the Approximation Methods**

Many of those methods require a lot of computing to come up with the GP approximation. Some of them require to find a set of $m << n$ inducing points representing the observations of the original GP. Once this step is done, they allow a very computationally efficient prediction ($\mathcal{O}(m^3)$). Unfortunately, we will see that the nature of the optimisation problem proposed by Krause et al. (2008), makes it impossible for us to use most of those algorithms. Indeed we will see that our observation set is constantly changing with the iterations of the algorithm, forcing us to compute for each iteration a new set of inducing points. The bottleneck would then be at the training of an approximated GP and not at its evaluation.

## 2.3 Covariance Matrix

We have seen how GPs are defined and can be made more scalable. In order to have good results, we need to have a good estimator of the covariance matrix between the locations in our simulation mesh.

In our specific case, we 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 to find optimal sensor locations. We can therefore have a **data-driven** approach to the covariance matrix estimation. We don't need an arbitrary kernel function and we can rely on the available data. Unfortunately, because of the number of locations considered we are also confronted to the challenge of estimating such a covariance matrix from samples.

We will see the properties that our covariance must have to accurately represent our data, before discussing the best ways of estimating this covariance matrix.

### 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 the stationarity.

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.

## 2.3.2 Covariance Estimation

There are several ways to estimate the covariance matrix of a random process. In this part, we are going to expose some simple, yet computationally efficient ways to estimate the true covariance matrix $\mathbf{\Sigma}$ of a gaussian distributed dataset.

We consider the process $Y$ in $p$ different locations at $n$ sampling times. $\boldsymbol{Y}_i = (Y_{1i}, \ldots, Y_{pi})$, with $i = 1 \ldots n$, and $X$ the cantered process.

### 2.3.2.1 Sample Covariance

The simplest estimator of the covariance matrix is the **sample covariance matrix** $\hat{\boldsymbol{S}}$ directly computed from the captured data. It's size if of $p \times p$. This estimator is an unbiased estimator of the true covariance matrix $\mathbf{\Sigma}$. It is also the expression of the **maximal likelihood** estimator.

$$\hat{\boldsymbol{S}} = \frac{1}{n} \sum_{i=1}^{n} (\boldsymbol{Y}_i - \bar{\boldsymbol{Y}}) \cdot (\boldsymbol{Y}_i - \bar{\boldsymbol{Y}})^T, \quad \bar{\boldsymbol{Y}} = \frac{1}{T} \sum_{i=1}^{n} \boldsymbol{Y}_i \tag{2.8}$$

$$\hat{\boldsymbol{S}} = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{X}_i \cdot \boldsymbol{X}_i^T \tag{2.9}$$

Unfortunately, this covariance is often singular when $p >> T$ (Fan et al., 2015), and its estimation error is very important (Ledoit and Wolf, 2003a). The sample covariance matrix can't be used directly in our optimisation algorithms.

### 2.3.2.2 Shrinkage Method

A good alternative to the sample covariance matrix is the **shrinkage method** developed by Ledoit and Wolf (2003b). It was originally proposed to improve the mean-variance portfolio optimisation in finance.

Intuitively, the algorithm tends to pull most extreme coefficients of the covariance matrix towards more central values, thus shifting every eigenvalue according to a given offset. The principle of shrinkage estimator is to make a compromise between two extreme estimators: One that is structured, that we call the **shrinkage target** $\boldsymbol{F}$, and one that is unstructured: the **sample covariance matrix** $\boldsymbol{S}$. To make this compromise between those, we take a convex linear combination of the two estimators and optimise the shrinkage constants. The shrinkage target $F$ is often defined as proportional to the identity matrix Chen et al. (2010):

$$\hat{\boldsymbol{F}} = \frac{\mathrm{Tr}(\hat{\boldsymbol{S}})}{p} \boldsymbol{I} \tag{2.10}$$

The shrinkage oracle estimator is defined such as:

$$\hat{\boldsymbol{\Sigma}} = \rho\hat{\boldsymbol{F}} + (1-\rho)\hat{\boldsymbol{S}} \tag{2.11}$$

#### 2.3.2.3 Ledoit-Wolf Shrinkage Estimator

An optimal choice for the parameter $\rho$ can be made. An optimisation problem was expressed as follows by Ledoit and Wolf (2004), ($\|\cdot\|_F$ being the Frobenius matrix norm ):

$$\min_{\rho} \quad \mathbb{E}\left\{ \|\hat{\boldsymbol{\Sigma}} - \boldsymbol{\Sigma}\|_F^2 \right\} \tag{2.12}$$

$$\text{s.t.} \quad \hat{\boldsymbol{\Sigma}} = \rho\hat{\boldsymbol{F}} + (1-\rho)\hat{\boldsymbol{S}} \tag{2.13}$$

$$\tag{2.14}$$

Which has for solution:

$$\rho_O = \frac{\mathbb{E}\left\{\text{Tr}[(\boldsymbol{\Sigma} - \hat{\boldsymbol{S}})(\hat{\boldsymbol{F}} - \hat{\boldsymbol{S}})]\right\}}{\mathbb{E}\left\{ \left\| \hat{\boldsymbol{S}} - \hat{\boldsymbol{S}} \right\|_F^2 \right\}} \tag{2.15}$$

This solution is dependent on the true covariance matrix $\boldsymbol{\Sigma}$ and therefore can't be directly implemented. Ledoit and Wolf (2004) propose then a method that is to *asymptotically* approximate the estimator.

$$\hat{\rho}^*_{LW} = \frac{\sum_{i=1}^{n} \left\| \boldsymbol{Y_i}\boldsymbol{Y_i}^T - \hat{\boldsymbol{S}} \right\|_F^2}{n^2 \left[ \text{Tr}(\hat{\boldsymbol{S}}^2) - \frac{\text{Tr}^2(\hat{\boldsymbol{S}})}{p} \right]} \tag{2.16}$$

They proved that for $n,\ p \to \infty$ and $p/n \to c$ with $0 < c < \infty$, this quantity converges to the optimal solution. By including it in the equation 2.11 we obtain the **Ledoit-Wolf (LW) covariance estimator**.

#### 2.3.2.4 Oracle Approximating Shrinkage (OAS)

An alternative proposed by Chen et al. (2010), is called the Oracle Approximating Shrinkage. It builds itself on the LW estimator, but with a smaller Mean Squared Error than the LW estimator.

By defining an iterative method to estimate he shrinkage constant, and by using the gaussian assumption, they have been able to develop a closed-form expression for the shrinkage constant $0 < \rho^*_{OAS} \leq 1$:

$$\rho^*_{OAS} = \min\left( \frac{\left(\frac{1-2}{p}\right)\text{Tr}(\hat{\boldsymbol{S}}^2) - \text{Tr}^2(\hat{\boldsymbol{S}})}{\left(\frac{n+1-2}{p}\right)\left[\text{Tr}(\hat{\boldsymbol{S}}^2) - \frac{\text{Tr}^2(\hat{\boldsymbol{S}})}{p}\right]} \quad , \ 1 \right) \tag{2.17}$$

Additionally, it is shown that in cases where $p$ is much larger than $n$, the OAS estimator performs better than the LW estimator.

### 2.3.3 Kernel Functions

One other alternative for estimating a covariance matrix it the choice of an arbitrary kernel function such as it is often the case in classical GP regressions. The kernel functions are often parametric and can, therefore, be adapted to the data, and optimised using the Maximal Likelihood of the GP regression.

Here, we display some classical kernel functions which are isotropic and have for parameter the lengthscale $l$. We present the squared **exponential kernel** and the **Matèrn 3/2** and **5/2** taken from Rasmussen and Williams (2006).

$$K_{SE}(d) = \exp\Big(-\frac{|d|^2}{2\ell^2}\Big) \tag{2.18}$$

$$K_{3/2}(d) = \sigma^2\left(1+\frac{\sqrt{3}d}{\rho}\right)\exp\left(-\frac{\sqrt{3}d}{\rho}\right) \tag{2.19}$$

$$K_{5/2}(d) = \sigma^2\left(1+\frac{\sqrt{5}d}{\rho}+\frac{5d^2}{3\rho^2}\right)\exp\left(-\frac{\sqrt{5}d}{\rho}\right) \tag{2.20}$$

As we have data for every point we wish to predict, it is better to use some data-driven technique rather than using some of those arbitrary covariance functions. Furthermore, the true covariance of the space is likely to be non-stationary as Krause et al. (2008) explains.

## 2.4 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 optimally 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 location of the mesh. In practice, this is not that obvious as we have seen in section 2.3. The following is valid for any covariance matrix that is both *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.

## 2.4.1 Placement Criterion

Here we define the criterion we will use to judge the quality of the placement. The most intuitive one being the Entropy, but not as accurate as the Mutual Information.

### 2.4.1.1 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):

$$H(\mathcal{X}_{\mathcal{V}\backslash\mathcal{A}}|\ \mathcal{X}_{\mathcal{A}}) = \mathbb{E}_{p(\boldsymbol{x}_{\mathcal{V}\backslash\mathcal{A}},\boldsymbol{x}_{\mathcal{A}})} \log p(\mathcal{X}_{\mathcal{V}\backslash\mathcal{A}}|\ \mathcal{X}_{\mathcal{A}}) \tag{2.21}$$

$$H(\mathcal{X}_{\mathcal{V}\backslash\mathcal{A}}|\ \mathcal{X}_{\mathcal{A}}) = -\sum_{\boldsymbol{x}_{\mathcal{V}\backslash\mathcal{A}}\in\mathcal{X}_{\mathcal{V}\backslash\mathcal{A}}} \sum_{\boldsymbol{x}_{\mathcal{A}}\in\mathcal{X}_{\mathcal{A}}} p(\boldsymbol{x}_{\mathcal{V}\backslash\mathcal{A}},\boldsymbol{x}_{\mathcal{A}}) \log p(\boldsymbol{x}_{\mathcal{V}\backslash\mathcal{A}}|\boldsymbol{x}_{\mathcal{A}}) \tag{2.22}$$

$$H(\mathcal{X}_{\mathcal{V}\backslash\mathcal{A}}|\ \mathcal{X}_{\mathcal{A}}) = -\int p(\boldsymbol{x}_{\mathcal{V}\backslash\mathcal{A}},\boldsymbol{x}_{\mathcal{A}}) \log p(\boldsymbol{x}_{\mathcal{V}\backslash\mathcal{A}}|\boldsymbol{x}_{\mathcal{A}})\, d\boldsymbol{x}_{\mathcal{V}\backslash\mathcal{A}}\, d\boldsymbol{x}_{\mathcal{A}} \tag{2.23}$$

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 conditional covariance between those elements: $K_{y|\mathcal{A}}$. Thus we have:

$$H(\mathcal{X}_y|\ \mathcal{X}_{\mathcal{A}}) = \frac{1}{2}\log K_{y|\mathcal{A}} + \frac{1}{2}(1+\log 2\pi) \tag{2.24}$$

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):

$$H(\mathcal{X}_{\mathcal{V}\backslash\mathcal{A}}|\ \mathcal{X}_{\mathcal{A}}) = H(\mathcal{X}_{\mathcal{V}\backslash\mathcal{A}},\ \mathcal{X}_{\mathcal{A}}) - H(\mathcal{X}_{\mathcal{A}}) \tag{2.25}$$

$$= H(\mathcal{X}_{\mathcal{V}}) - H(\mathcal{X}_{\mathcal{A}}) \tag{2.26}$$

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 $\mathcal{V}\backslash\mathcal{A}$:

$$\mathcal{A}^* = \arg\min_{\mathcal{A}\subseteq\mathcal{V}:|\mathcal{A}|=k} H(\mathcal{X}_{\mathcal{V}\backslash\mathcal{A}}|\ \mathcal{X}_{\mathcal{A}}) \tag{2.27}$$

$$= \arg\min_{\mathcal{A}\subseteq\mathcal{V}:|\mathcal{A}|=k} H(\mathcal{X}_{\mathcal{V}}) - H(\mathcal{X}_{\mathcal{A}}) \tag{2.28}$$

$$= \arg\max_{\mathcal{A}\subseteq\mathcal{V}:|\mathcal{A}|=k} H(\mathcal{X}_{\mathcal{A}}) \tag{2.29}$$

#### 2.4.1.2 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):

$$I(\mathcal{X}_{\mathcal{V}\backslash\mathcal{A}},\ \mathcal{X}_{\mathcal{A}}) = H(\mathcal{X}_{\mathcal{V}\backslash\mathcal{A}}) - H(\mathcal{X}_{\mathcal{V}\backslash\mathcal{A}}|\mathcal{X}_{\mathcal{A}}) \tag{2.30}$$

We can redefine our problem as the maximisation of the Mutual Information between the set of selected sensors $\mathcal{A}$ and the rest of the space $\mathcal{V}\backslash\mathcal{A}$:

$$\mathcal{A}^* = \arg\max_{\mathcal{A}\subseteq\mathcal{V}:|\mathcal{A}|=k} I(\mathcal{X}_{\mathcal{V}\backslash\mathcal{A}},\ \mathcal{X}_{\mathcal{A}}) \tag{2.31}$$

$$= \arg\max_{\mathcal{A}\subseteq\mathcal{V}:|\mathcal{A}|=k} H(\mathcal{X}_{\mathcal{V}\backslash\mathcal{A}}) - H(\mathcal{X}_{\mathcal{V}\backslash\mathcal{A}}|\mathcal{X}_{\mathcal{A}}) \tag{2.32}$$

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 2.3) for giving good results.

### 2.4.2 Approximation Algorithm

As stated by Krause et al. (2008), the problem is a *NP-complete problem*. Therefore we present here an algorithm that can 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:

$$y* = \arg\max_{y\in\mathcal{S}\backslash A} MI(\mathcal{A}\cup y, \bar{\mathcal{A}}) - MI(\mathcal{A}, \mathcal{V}\backslash\mathcal{A}) \tag{2.33}$$

$$= \arg\max_{y\in\mathcal{S}\backslash A} H(y|\mathcal{A}) - H(y|\bar{\mathcal{A}}) \tag{2.34}$$

In our specific case with the Multivariate Gaussian Distribution, we can take the formulation presented in equation 2.24 and rewrite the objective as:

$$y* = \arg\max_{y \in \mathcal{S} \setminus A} \frac{1}{2} \log K_{y|\mathcal{A}} - \frac{1}{2} \log K_{y|\bar{\mathcal{A}}} \tag{2.35}$$

$$= \arg\max_{y \in \mathcal{S} \setminus A} \log \left( \frac{K_{y|\mathcal{A}}}{K_{y|\bar{\mathcal{A}}}} \right) \tag{2.36}$$

$$y* = \arg\max_{y \in \mathcal{S} \setminus A} \frac{K_{y|\mathcal{A}}}{K_{y|\bar{\mathcal{A}}}} \tag{2.37}$$

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:

$$y* = \arg\max_{y \in \mathcal{S} \setminus A} \frac{K_{yy} - K_{y\mathcal{A}} K_{\mathcal{A}\mathcal{A}}^{-1} K_{\mathcal{A}y}}{K_{yy} - K_{y\bar{\mathcal{A}}} K_{\bar{\mathcal{A}}\bar{\mathcal{A}}}^{-1} K_{\bar{\mathcal{A}}y}} \tag{2.38}$$

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** $y \in \mathcal{S} \setminus \mathcal{A}$ **do**
    $\delta_y \leftarrow \frac{K_{yy} - K_{y\mathcal{A}} K_{\mathcal{A}\mathcal{A}}^{-1} K_{\mathcal{A}y}}{K_{yy} - K_{y\bar{\mathcal{A}}} K_{\bar{\mathcal{A}}\bar{\mathcal{A}}}^{-1} K_{\bar{\mathcal{A}}y}}$
  **end**
  $y^* \leftarrow \arg\max_{y \in \mathcal{S} \setminus A} \delta_y$
  $\mathcal{A} \leftarrow \mathcal{A} \cup y^*$
**end**

**Algorithm 1:** Greedy/Naive 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:

$$MI(\hat{\mathcal{A}}) \geq (1 - \frac{1}{e}) \cdot MI(\mathcal{A}^*) - k\epsilon \tag{2.39}$$

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.

### 2.4.3 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 $\mathcal{O}(kn)$

#### 2.4.3.1 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 $\delta_y$ and will, therefore, be less desirable. Therefore if we maintain a priority queue with the ordered values of $\delta_y$ 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 $\delta_y$ 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 $K_{\mathcal{VV}}$ , $k$, $\mathcal{V}$
**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**
    $y^* \leftarrow \arg\max_{y \in \mathcal{S} \backslash A} \delta_y$
    **if** $current_{y^*}$ **then** break;
    $\delta_{y^*}$ is updated with 2.38
    $current_y \leftarrow$ True
  **end**
  $\mathcal{A} \leftarrow \mathcal{A} \cup y^*$
**end**

**Algorithm 2:** Lazy Algorithm

#### 2.4.3.2 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. This method is equivalent to using sparse kernels for estimating the covariance matrix.

**Data:** Covariance matrix $K_{\mathcal{VV}}$ , $k$, $\mathcal{V}$, $\epsilon$
**Result:** Sensor Selection $\mathcal{A}$
initialisation;
$\mathcal{A} = \emptyset$
**foreach** $y \in \mathcal{S}$ **do** $\delta_y \leftarrow$ 2.42 ;
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** $\delta_y \leftarrow$ 2.44 ;
  **end**
**end**

**Algorithm 3:** Local Kernel Algorithm

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 $\delta_y$ values are initialised by taking the difference between the true entropy and the truncated covariance, as:

$$\delta_y = \tilde{H}_\epsilon(y|\mathcal{A}) - \tilde{H}_\epsilon(y|\bar{\mathcal{A}}) \tag{2.40}$$
$$= H(y) - \tilde{H}_\epsilon(y|\mathcal{V}\backslash y) \tag{2.41}$$

Or equivalently by keeping the previous notations:

$$\delta_y = \frac{K_{yy}}{K_{yy} - \tilde{K}_{y\mathcal{V}\backslash y} \tilde{K}^{-1}_{\mathcal{V}\backslash y\mathcal{V}\backslash y} \tilde{K}_{\mathcal{V}\backslash yy}} \tag{2.42}$$

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

$$\delta_y = \tilde{H}_\epsilon(y|\mathcal{A}) - \tilde{H}_\epsilon(y|\bar{\mathcal{A}}) \tag{2.43}$$
$$= \frac{K_{yy} - \tilde{K}_{y\mathcal{A}} \tilde{K}^{-1}_{\mathcal{A}\mathcal{A}} \tilde{K}_{\mathcal{A}y}}{\tilde{K}_{yy} - \tilde{K}_{y\bar{\mathcal{A}}} \tilde{K}^{-1}_{\bar{\mathcal{A}}\bar{\mathcal{A}}} \tilde{K}_{\bar{\mathcal{A}}y}} \tag{2.44}$$

Furthermore, we also decide to update only the points that are not very correlated with the current optimal (previously placed sensor). This is justified by the fact that correlated points share a lot of mutual information and that the next best point is likely to be outside of $N(y^*, \epsilon)$. We explicitly define all the steps in algorithm 3.

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.

# Chapter 3

# Implementation

In this chapter, we are going to show details of the implementation. First, we give some practical details on the code and the tools used to create it. We will then proceed to the analysis of the data used in this project. We will explain the important role of data preselection, before showing details of the optimisation implementation. Finally, we will explain our choices for the application involving Data Assimilation.

## 3.1 Practical Informations

### 3.1.1 Availability of the code

All the codes developed during this project are available on **GitHub** at the following address: `https://github.com/adrianlwn/MScProject_OptSensors_GP`. A Description of the code and explanation on how to use it is available in appendix B.

### 3.1.2 Environment

For this project the code was implemented using *python 3.7* and the following list of external libraries:

| Library | Version |
|---|---|
| numpy | 1.16.2 |
| scipy | 1.2.1 |
| scikit-learn | 0.20.3 |
| pandas | 0.24.2 |
| shapely | 1.6.4 |
| matplotlib | 3.0.3 |
| fluidity | 4.1.15 |

**Table 3.1:** Library Informations

### 3.1.3 Experimental Conditions

All the implementation and the results obtained, especially the timings, have been computed on the same hardware and in the same conditions. The specifications of the machine are as follows:

| | |
|---|---|
| OS | Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-46-generic ) |
| Architecture | x86_64 |
| Number of CPUs | 48 |
| CPU | Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz |

**Table 3.2:** Hardware and Software Informations

## 3.2 Data Analysis

The main dataset we are using for this project comes from the air pollution simulations at the London South Bank University (LSBU) that was used by Arcucci et al. (2019). Those simulations were performed using the fluid dynamics model **Fluidity**.

The simulation results used are stored in Visualization Toolkit (VTK) files, and extracted using the python framework of **Fluidity**. We specifically use the data from the three-dimensional `small3DLSBU` simulation. We have developed a series of functions for reading, processing, and saving the raw simulations files in the project.
The simulated fields are the **tracer** concentration, **tracer background** concentration, wind **velocity** and **pressure**.

The simulation is done on a discrete set of points, each having a 3D coordinate and organised as an unstructured mesh. In this mesh there is a total of $100'040$ locations, covering a volume of $720m \times 680m \times 250m$. At the low centre of the space, the density of the points is very high compared to the rest of the space. Furthermore, the simulation is realised at a time resolution of $\Delta t = 0.5s$ for $988$ time steps or samples.

### 3.2.1 Tracer Concentration Data

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. We represent two horizontal cuts of the tracer field at $z = 1m$ and at $t = 50$ and $t = 988$ in Figure 3.1

As we can see by observing the time propagation of those fields, the wind is pushing the pollutants in the east direction. Because of this, we observe that the **tracer** concentration is mainly visible downwind.

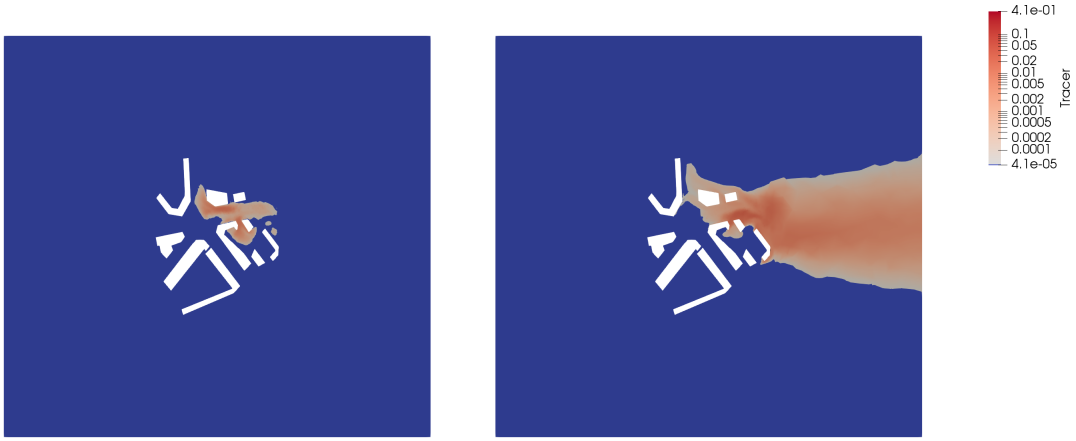


**Figure 3.1:** Tracer Field: Horizontal cut at $z = 1m$ and $t = 50$ / $t = 988$

We can also visualise the number of zero elements present in the data *in function of the time*. We count every point of data that has a value superior $\tau = 10^{-12}$. This gives us the plot of Figure 3.2. As we can see, it takes some initial time for the tracer to propagate to some kind of steady-state, at around 60% of the points.

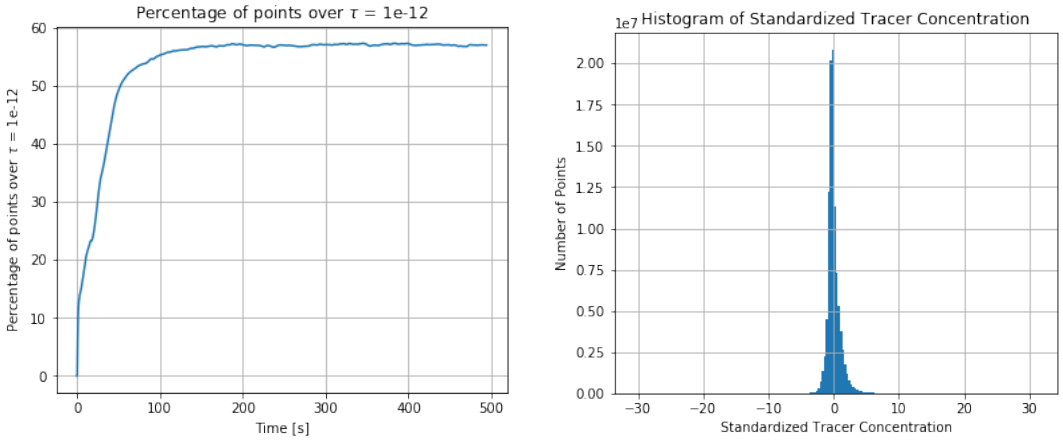


**Figure 3.2:** Percentage of significant points in function of the time for $\tau = 10^{-12}$ & Histogram of the Standardised Tracer Data

Finally we visualise the histogram distribution of all the *Tracer* values (time and space) in the Figure 3.2. We have **standardised** the data at each location to verify if the distributions are Gaussian. As we can see the distribution seems to be close to gaussian and therefore fulfils the conditions of application of the algorithm.

### 3.2.2 Pressure Data

We then focus on the Pressure Data present in our simulation.

We plot the field at times $t = 50$ and $t = 988$ in the Figure 3.3. There is no obvious trend linked to the propagation of the fluid. We also decide to make a histogram of the data (without standardisation) and obtain the Figure 3.4. Similarly, we can say that the *Pressure* data seems to have a Gaussian distribution.


**Figure 3.3:** Pressure Field: Horizontal cut at $z = 1m$ and $t = 50$ / $t = 988$

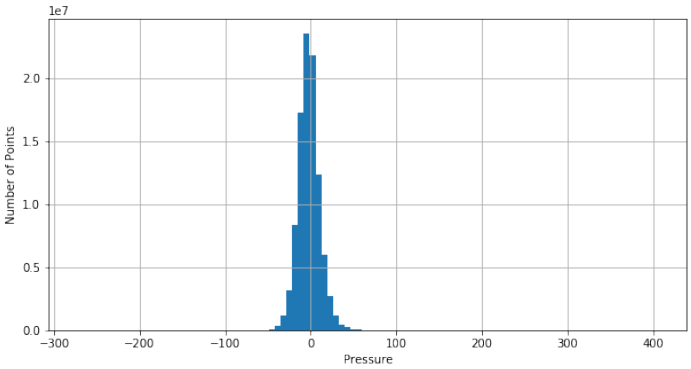


**Figure 3.4:** Histogram of Pressure Values

## 3.3 Preselection of the Data

As we have seen previously there is a large number of irrelevant data points inside the original dataset. A lot of points have a tracer concentration close to $0$ and most of the relevant points are found downwind of the origin. In this section, we are going to see how to remove irrelevant points and at the same time make the dataset smaller and therefore reduce the computational cost of the main optimisation problem.

### 3.3.1 Selection of a working subset of the data

The first approach that we take to reduce the number of potential sensor locations, is the reduction of the space in which we run our optimisation problem. We are

focusing on the *tracer* field of the simulation data. This field contains the propagation of a pollutant originating at the **center of the space** and under wind conditions blowing in the *east direction* (see Figure 3.1 ). The resulting data shows that most of the space is unaffected by this pollutant and we, therefore, wish to select only the space in which the pollutant concentration is non-negligible. For that, we develop the following procedure.

First, we cut our 3D space into cuboids. For that we fix a number of bins per dimension and we obtained $R$ subsets $\{\mathcal{D}_k\}_{k=1...R}$. For each subspace we compute the sum over time and space of the tracer values $Y_t^i$ :

$$C(\mathcal{D}_i) = \sum_{k \in \mathcal{D}_i} \sum_{t=0}^{T} Y_t^k \tag{3.1}$$

We apply then a selection of the subsets based on the value of $C(\mathcal{D}_i)$ and a threshold $\tau$. We keep then every subset $\mathcal{D}_i$ that respects the condition: $C(\mathcal{D}_i) > \tau$. This condition guarantees that the points kept in the new working subset $S$ have sufficient importance in the physical space. The new working subset for our optimisation problem would then be:

$$S = \bigcup_{C(\mathcal{D}_i)>\tau} \mathcal{D}_i \tag{3.2}$$

An illustration of this algorithm is now detailed. For a number of bins of $25$ per dimension and a threshold of $\tau = 10^{-2}$, we obtain a new working set size $|S| = 57'725$ instead of an original number of $100'040$. It is displayed on the Figure 3.5

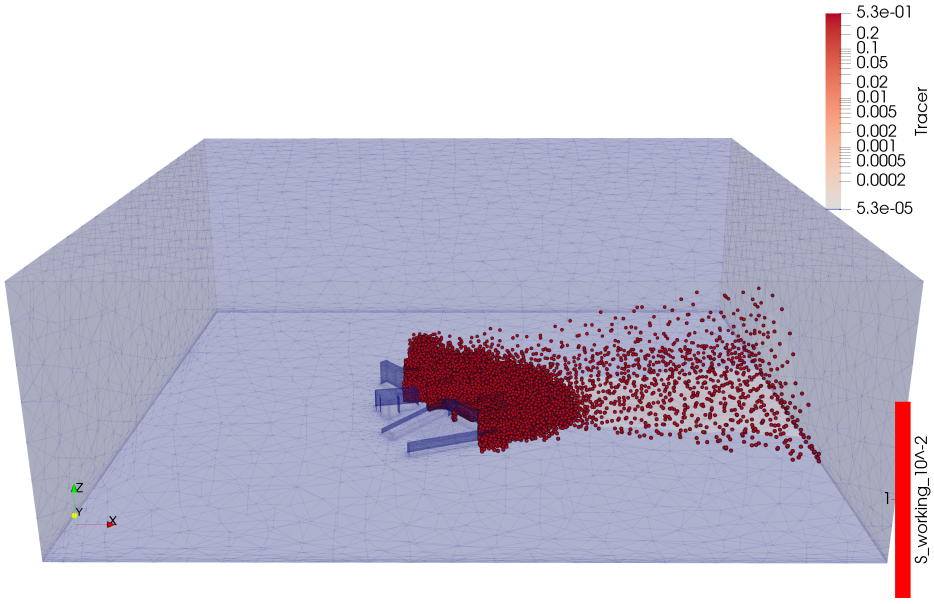


**Figure 3.5:** New working subset $S$ for $\tau = 10^{-2}$

### 3.3.2 Selection of a subset at human level

To further reduce the number of points involved in the optimisation problem, we can also consider taking points that are only **accessible by a human from the ground level or the buildings**. This makes sense in the way that sensors need to be reached from the ground and the building tops and sides for their initial placement and maintenance.

We define for each building $i \in \{1, \ldots, B\}$ an altitude $H_i$, and for $i = 0$ we consider the rest of the unoccupied space, so $H_0 = 0$. This allows us to define an altitude under which we will select the points: $H_i + h$. The area covered by the buildings is enlarged by the value $w$, so that is covers also the sides of the buildings. See the illustration provided in Figure 3.6.

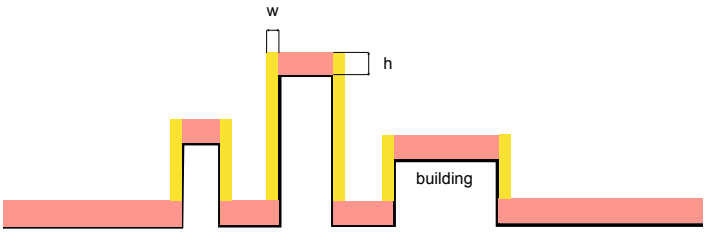


**Figure 3.6:** Chart Presenting the Building Profile in Human Level Selection

In order to proceed to this selection, I had to overcome the absence of defined building profile and I had to manually define the shapes of the buildings (as you can see on Figure 3.7) by measuring empirically the coordinates of the buildings in the small LSBU dataset, considering an XY projection of the 3D space.

**Figure 3.7:** Empirical Building Shapes

Once those coordinates acquired, we used the **shapely** library, to define the polygons associated with the buildings. To define the height $H_i$ of each building, we had to define a set of points overhanging it and find the minimal altitude of those points. To define the set of points I took the inner part of the surface of the building (cut by 1m) to avoid any edge point. This can be seen in the Figure 3.8). The points

which have XY coordinates within this shape are then selected and the minimal Z coordinate is taken as the definition of the roof level $H_i$

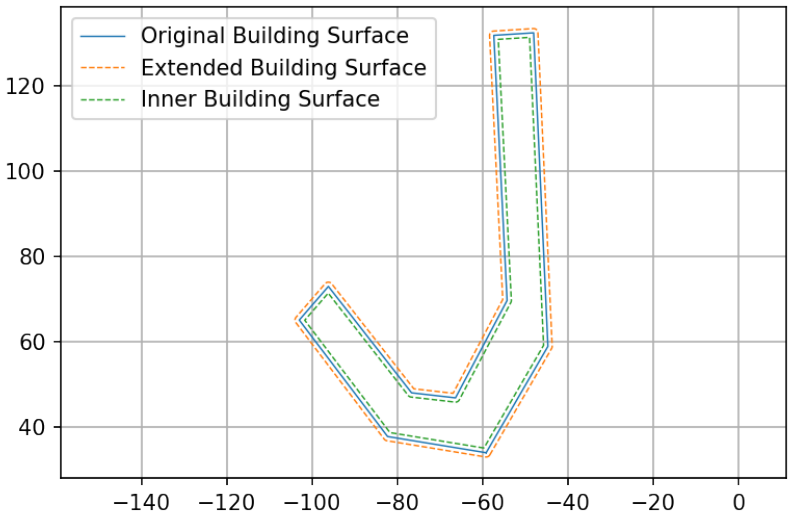


**Figure 3.8:** Extended an Inner building surface

Once the roof level defined we come back to the original shape of the building and enlarge it of $w$ such as shown in Figure 3.8. We then select every point of the main dataset that has XY coordinates in this extended rooftop and choose only the ones which have altitude Z bellow the threshold $H_i + h$. This constitutes our human level data selection. An illustration can be found on Figure 3.9.

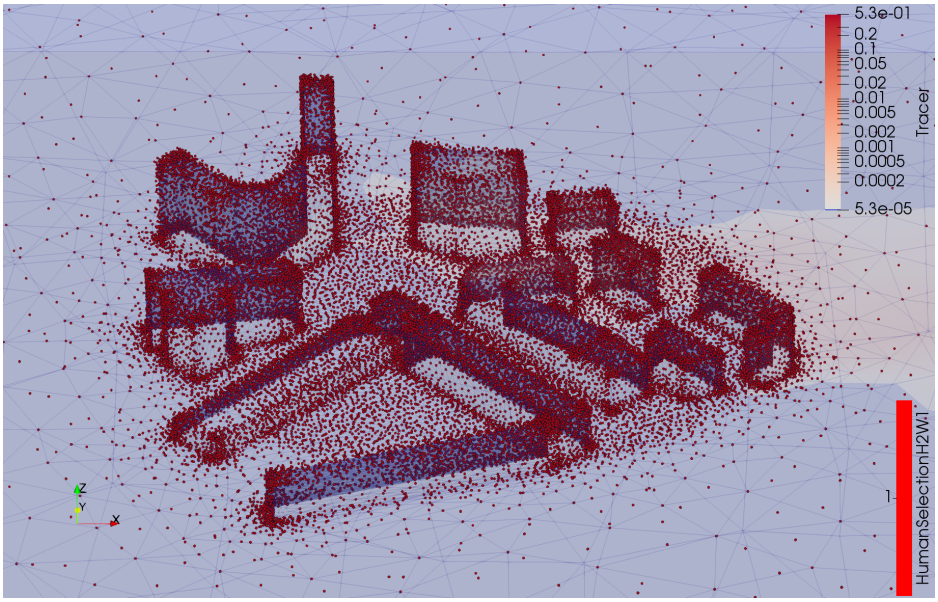


**Figure 3.9:** Human Level Selection

This method can be seen as quite empirical, but it enables a precise dataset selection

with no outlying irrelevant point.

With the parameters $h = 2\text{m}$ and $w = 1\text{m}$ we reduce the size of the dataset to $|S| = 37'847$.

### 3.3.3 Combined Selection

By combining the two selection approaches and by taking the intersection of the two previously defined datasets: the working subset based on the values of the tracer and the human level selection. We can reduce the number of points in the dataset to $|S| = 23'643$, instead of an original number of $100'040$, which is a reduction of $76.36\%$ of the original size. In Figure 3.10 we see an illustration of the selected dataset that will be used in the rest of the project.

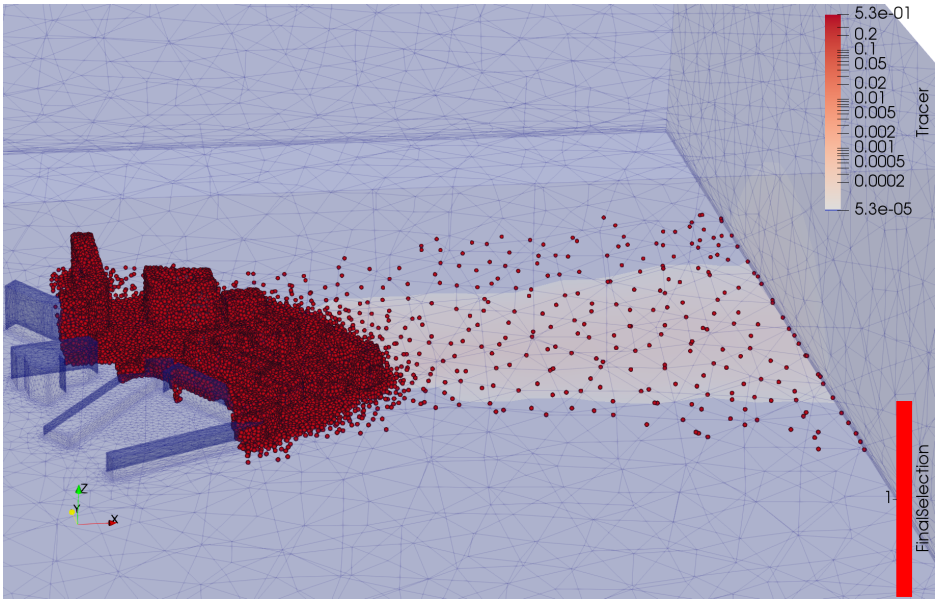


**Figure 3.10:** Combined Selection: Working Subset and Human Level Selection

## 3.4 Implementation of Covariance Matrix

After the pre-selection of the data, we proceed to the computation of the covariance matrix needed for the optimisation process. This covariance links each point of the space with each other, having the size $23'643 \times 23'643$.

We take advantage of the *sklearn* library that contains a number of useful methods including one computing the Sample Covariance, the LW and the OAS shrinkage estimators.

## 3.5 Implementation of Gaussian Processes

The GPs are a central concept used in this project. We will cover here how they were implemented and what kind of approximation was implemented in order to make them more scalable.

### 3.5.1 Classical GPs

As we have seen in section 2.2, the Gaussian Process conditional covariance estimation requires only the knowledge of the covariance matrix $\mathcal{K}$.

$$\boldsymbol{K}_{y|\mathcal{A}} = \boldsymbol{K}_{yy} - \boldsymbol{K}_{y\mathcal{A}}\boldsymbol{K}_{\mathcal{A}\mathcal{A}}^{-1}\boldsymbol{K}_{\mathcal{A}y} \tag{3.3}$$

The first approach is simply to implement this equation using the *numpy* library. It has the advantage to use parallel processing that exploits the multiple CPUs of our server, to speed up any linear algebra computation, such as this one.

### 3.5.2 GPs Approximation with TSVD

In GPs, the inversion of the matrix $\mathcal{K}_{\mathcal{A}\mathcal{A}}$ is the most expensive operation, as it has a complexity of $\mathcal{O}(n^3)$. In to reduce the computational cost of the Gaussian Process, we propose a method that allows the approximation of the covariance by reducing the dimension of the data, using the **Truncated Singular Values Decomposition** (TSVD). The idea has been developed by (Hansen, 1987) as it allows to regularise matrix in ill-posed problems.

As defined previously, the centred data is $\boldsymbol{X}$ and the sample covariance matrix is $\hat{\boldsymbol{S}}$. The data $\boldsymbol{X}$ has for size $p \times n$ and we would like to approximate it by a matrix of size $\tau \times n$. The singular value decomposition (SVD) allows to express the data matrix as a product of a rectangular diagonal matrix $\boldsymbol{\Theta} = \text{diag}(\sigma_1, \ldots, \sigma_n) \in \mathbb{R}^{p\times n}$ containing the singular values, and two orthogonal singular vectors matrices $\boldsymbol{V} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{U} \in \mathbb{R}^{p\times p}$, such as:

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Theta}\boldsymbol{V}^T \tag{3.4}$$

We truncate the SVD from $p$ to $\tau$ by keeping only the $\tau$ largest singular values. We have the diagonal matrix $\boldsymbol{\Theta}_\tau = \text{diag}(\sigma_1, \ldots, \sigma_\tau, 0, \ldots, 0) \in \mathbb{R}^{p\times n}$. We are then able to express an approximation of the data $\tilde{\boldsymbol{X}}$:

$$\tilde{\boldsymbol{X}}_\tau = \boldsymbol{U}\boldsymbol{\Theta}_\tau\boldsymbol{V}^T \tag{3.5}$$

We apply this approach to the set of points $\mathcal{A}$ with the data $\boldsymbol{X}_\mathcal{A}$. We get the resulting TSVD for this data: $\tilde{\boldsymbol{X}}_{\mathcal{A},\tau} \in \mathbb{R}^{\tau\times n}$ we can compute with it an approximation of the sample covariance that can be easily inverted and used in our GP based optimisation.

$$\tilde{\boldsymbol{K}}_{\mathcal{A}\mathcal{A}} = \frac{1}{n}\tilde{\boldsymbol{X}}_{\mathcal{A},\tau}\tilde{\boldsymbol{X}}_{\mathcal{A},\tau}^T \tag{3.6}$$

We also replace the vector of covariance $\boldsymbol{K}_{y\mathcal{A}}$ by:

$$\tilde{\boldsymbol{K}}_{y\mathcal{A}} = \frac{1}{n}\boldsymbol{X}_y\tilde{\boldsymbol{X}}_{\mathcal{A},\tau}^T \tag{3.7}$$

We end up with an expression of TSVD approximate GP variance:

$$\tilde{\boldsymbol{K}}_{y|\mathcal{A}} = \frac{1}{n}\left(\boldsymbol{X}_y\boldsymbol{X}_y^T - \boldsymbol{X}_y\tilde{\boldsymbol{X}}_{\mathcal{A},\tau}^T\left(\tilde{\boldsymbol{X}}_{\mathcal{A},\tau}\tilde{\boldsymbol{X}}_{\mathcal{A},\tau}^T\right)^{-1}\tilde{\boldsymbol{X}}_{\mathcal{A},\tau}\boldsymbol{X}_y^T\right) \tag{3.8}$$

With this approximation, the inversion has a complexity reduced to $\mathcal{O}(\tau^3)$, which is a great improvement. It has the advantage not to require any estimation of the true covariance matrix. The TSVD acts as a regulariser and the sample covariance taken on the reduced data is symmetric and positive definite.

However, this method requires the computation of a new TSVD at each iteration of the algorithm as the indexes in $\mathcal{A}$ are constantly changing. This has a cost but it is much faster than applying any other GP approximation method.

### 3.5.3 Limitations of Other Approaches

A lot of literature has been studied on how GPs could be made more scalable: approaches relying on very elegant solutions, such as sparse GPs and VFE. After implementing them using specialised GP library (such as *GPy*), we discovered that they could not be applied to our optimisation problem. Sparse GPs are in fact optimisation problems that allow the selection of a reduced number of positions to represent a large set of observations. But in our algorithm, the observation set changes at every step of the loop and we, therefore, need a new approximation for predicting at only one single point. This makes this approach computationally inefficient for the algorithms of Krause et al. (2008).

If we analyse the algorithm 1 in more details, we see that it is difficult to implement a method to approximate the GP at each step.

We have defined $\mathcal{A}$ as the set of already placed sensors, and $\bar{\mathcal{A}}$ the set of other locations: $\bar{\mathcal{A}} = \mathcal{V}\backslash\{\mathcal{A} \cup y\}$ and $y$ is the candidate point changing at each step. We then have the full set of locations that is cut in 3 parts: $\mathcal{V} = \mathcal{A} \cup \bar{\mathcal{A}} \cup \{y\}$

At each iteration, we need to compute two different GPs. The first one is computing the conditional variance for the point $y$ knowing the set of points $\mathcal{A}$: $\boldsymbol{K}_{y|\mathcal{A}} = \boldsymbol{K}_{yy} - \boldsymbol{K}_{y\mathcal{A}}\boldsymbol{K}_{\mathcal{A}\mathcal{A}}^{-1}\boldsymbol{K}_{\mathcal{A}y}$. This GP is quite simple to compute as our cardinality of $\mathcal{A}$ is small. Moreover, the covariance matrix $\boldsymbol{K}_{\mathcal{A}\mathcal{A}}$is not changing through the 2nd loop as $\mathcal{A}$ stays the same, so the inversion needs only to be done once.

The second GP is computing the conditional variance for the point y knowing the set of points $\bar{\mathcal{A}}$: $\boldsymbol{K}_{y|\bar{\mathcal{A}}} = \boldsymbol{K}_{yy} - \boldsymbol{K}_{y\bar{\mathcal{A}}}\boldsymbol{K}_{\bar{\mathcal{A}}\bar{\mathcal{A}}}^{-1}\boldsymbol{K}_{\bar{\mathcal{A}}y}$. This is much more difficult to estimate when the number of points in $\bar{\mathcal{A}}$ is of the order of $100'000$. Not only it is difficult to estimate once, but be have to to it for every step of the second loop as the set $\bar{\mathcal{A}}$

changes constantly as we move the candidate point $y$. We then have a covariance matrix $\boldsymbol{K}_{\bar{A}\bar{A}}$ that is fundamentally different between each step.

Because of this, if we want to use a method that allows the removal of the bottleneck at the covariance matrix inversion, we need to avoid creating another bottleneck to approximate the GP. The TSVD approach is sufficiently light not to increase too much the computational cost. But other methods, such as sparse GPs are not usable with this algorithm.

## 3.6 Implementation of the Optimisation

Here we review the implementation details of the optimisation algorithm itself and its variations. We are going to optimally find a number of $k = 10$ sensors in the LSBU dataset.

### 3.6.1 Optimisation Algorithms

#### Greedy Algorithm

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:

$$y^* = \arg\max_{y \in \mathcal{S}\backslash A} \frac{K_{y|\mathcal{A}}}{K_{y|\bar{\mathcal{A}}}} \tag{3.9}$$

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. In this version is directly implemented using a double for loop. The first loop is iterating on the $k$ sensors to place, and the inner loop in iterating through all the candidates.

#### Lazy Algorithm

The Lazy version of the algorithm (Algorithm 2) uses the mathematical concept of sub-modularity. This allows for reducing drastically the number of candidate points to update. By using the python `heapq`, a *min-heap* binary structure, to store the mutual information improvements, we can maintain the results sorted as we update each candidate. By definition this structure sorts the elements from the smallest to the largest, we store the results we have added a minus sign, making this equivalent to a *max-heap*. This structure is highly efficient and allows to find the maximum in $\mathcal{O}(1)$ and to update a value in $\mathcal{O}(\log n)$.

**Local Kernel Algorithm**

The Local Kernel version of the algorithm (Algorithm 3) relies on the same greedy approach than the first algorithm. Therefore the implementation is very similar. The main difference relies on the use of a reduced set for the computation of the GPs and the set on which there is no update of the candidate points.

The number of points involved in the GPs $|N(y, \epsilon)| \leq d$ depends on the threshold on the covariance $\epsilon$. This is a parameter to optimise to make the computational time acceptable while retaining accuracy. It highly depends on the data and the covariance matrix entries.

### 3.6.2 Comparison Tool: Distance Between Sets

To be able to compare the results of optimisations, we define a useful metric that can be used to measure how similar are two optimal sets.

The output of our optimisation problem is a set of points $\mathcal{A}$ of size $k$, indexed such as $\mathcal{A} = \{a_1, \ldots, a_k\}$. We consider two optimal sets given by two versions of our algorithm: $\mathcal{A}_1$ and $\mathcal{A}_2$. We would like to define a metric that measures the distance between those two sets. As we will see several options are possible.

We could average the coordinates of the set's points and take the $l2$ distance between the averages ($\boldsymbol{x}(\cdot)$ being the coordinate of a point and $\bar{\boldsymbol{x}}(\mathcal{A})$ the average on a set $\mathcal{A}$ ):

$$d_{av}(\mathcal{A}_1, \mathcal{A}_2) = \| \bar{\boldsymbol{x}}(\mathcal{A}_1)] - \bar{\boldsymbol{x}}(\mathcal{A}_1) \|_2 \tag{3.10}$$

This is unlikely to measure the spread of the points and how each point is close to the other set. This is why we propose to create a **nearest neighbour** distance between the points of the sets.

First, we define a divergence between the sets $\mathcal{A}_1$ and $\mathcal{A}_2$, which takes for each point of $\mathcal{A}_1$ its closest neighbour in $\mathcal{A}_2$, takes the distance between them, and average it across the set. We define a nearest neighbour function $NN(\cdot, \cdot)$ which, given an index $a_i$ in the set $\mathcal{A}_1$ and the set $\mathcal{A}_2$, finds the index in the set $\mathcal{A}_2$ of closest points to $a_i$. This mapping is not bijective and therefore leads to asymmetrical results.

$$div_{NN}(\mathcal{A}_1, \mathcal{A}_2) = \frac{1}{k} \sum_{i=1}^{k} \| \boldsymbol{x}(a_i) - \boldsymbol{x}(NN(a_i, \mathcal{A}_2)) \|_2 \tag{3.11}$$

This metric has for unit the meter $[m]$. We also call this metric a **divergence** as it not *symmetric*. In order to define a distance that is symmetric, we define as follows the **nearest neighbour distance**:

$$d_{NN}(\mathcal{A}_1, \mathcal{A}_2) = \frac{1}{2}(div_{NN}(\mathcal{A}_1, \mathcal{A}_2) + div_{NN}(\mathcal{A}_2, \mathcal{A}_1)) \tag{3.12}$$

This metric has the advantage of being *symmetric*, taking into account the distance from each point of the dataset and therefore give a meaningful interpretation of how close are spatially the sets from each other.

## 3.7 Data Assimilation

Data Assimilation (DA) is a method widely used in the Fluid Simulations and Meteorology. Its purpose is to incorporate real observations in the simulation of a complex physical phenomenon. Variational Data Assimilation is an optimisation problem that allows an optimal assimilation of measurements to a simulation.

In the MAGIC project VarDA is a way of improving the light-simulation of the tracer concentration propagation through space. For this purpose points to be assimilated have to be selected. It is therefore interesting to apply the optimal sensor positioning algorithm to this simulated dataset, in order to select a limited number of points before applying the DA. For the implementation of this application, we rely mainly on the work of Arcucci et al. (2019). We use a code solving the VarDA problem with the L-BFGS (Limited – Broyden Fletcher Goldfarb Shanno) Algorithm.

The only modification that was done to the DA implementation was the removal of the TSVD of the Deviation Matrix. We don't need any space-reduction method as we have a small number of points to assimilate ($k = 10$).

To verify the accuracy of the DA procedure, we compute the Mean Square Error (MSE) between the actual state $y$ and the state following the assimilation $x_{DA}$. And we compare it to the background MSE between $y$ and the background state $x_B$. We will compare the results for the optimal sets to a series of randomised points.

# Chapter 4

# Analysis and Results

After analysing the dataset and implementing the optimisation algorithms, we have run several experiments to find the near-optimal points. For that, we have explored different approaches for estimating the covariance function; we have compared the different algorithm on a subset of the data; we have run the optimisation on the full dataset; and finally, we have applied those results to a Data Assimilation Procedure.

## 4.1 Covariance

As we have seen, our approach relies heavily on a properly defined covariance estimation. We have applied several methods to find a good covariance and we are presenting them in this section.

From now on, we will consider that the full dataset is our pre-selected subset with $23'643$ locations.

### 4.1.1 Sample Covariance

The simplest covariance estimation is the Sample (Empirical) Covariance. This Matrix is a very poor estimate, as it is singular and not positive definite and poorly conditioned. It makes it impossible to invert to use it for the GPs of our optimisation problem.

We compute the determinant of the matrix using `numpy.linalg.slogdet` for the stability of this function. It indicates us that the determinant is clearly negative, and the logarithm of the determinant's absolute value is: $-1'399'936.286$. The largest eigenvalue is: $\lambda_1 = 0.0219$ and the smallest eigenvalue is: $\lambda_p = -6.8929 \cdot 10^{-18}$.
We plot in Figure 4.1 the eigen-decomposition of this sample covariance matrix.
So the sample covariance matrix can not be directly used for our optimisation algorithm.

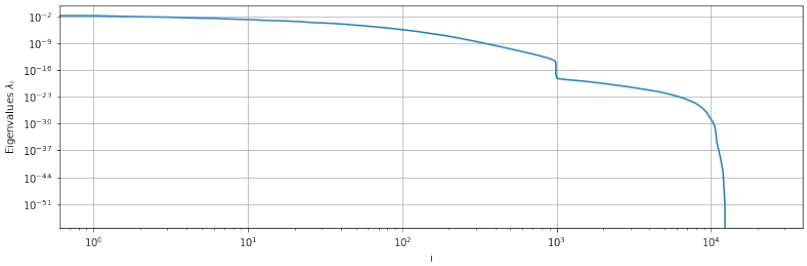


**Figure 4.1:** Empirical Covariance: Eigenvalues - Tracer

### 4.1.2 Shrinkage Covariance

In this section we show the results of the class of shrinkage covariance estimators for the *Tracer* Data. We use a simple **shrinkage** with shrinkage constant $\rho = 0.1$, $\rho = 0.5$ and $\rho = 0.8$, the **Ledoit-Wolf** Shrinkage and the **OAS** Shrinkage.

| | SignDet | LogDet | $\lambda_1$ | $\lambda_p$ | $\rho$ |
|---|---|---|---|---|---|
| Empirical | $(-)$ | $-1'399'936.28$ | $0.0219$ | $-6.8929 \cdot 10^{-18}$ | - |
| Shrinkage $\rho = 0.1$ | $(+)$ | $-351'807.64$ | $0.0197$ | $3.3603 \cdot 10^{-7}$ | $0.1$ |
| Shrinkage $\rho = 0.5$ | $(+)$ | $-314'035.58$ | $0.0109$ | $1.6801 \cdot 10^{-6}$ | $0.5$ |
| Shrinkage $\rho = 0.8$ | $(+)$ | $-303'046.20$ | $0.00438$ | $2.6882 \cdot 10^{-6}$ | $0.8$ |
| Ledoit-Wolf | $(+)$ | $-406'320.19$ | $0.0217$ | $3.2867 \cdot 10^{-8}$ | $0.0097811$ |
| OAS | $(+)$ | $-408'903.33$ | $0.0217$ | $2.9435 \cdot 10^{-8}$ | $0.0087595$ |

**Table 4.1:** Shrinkage Comparison - Tracer

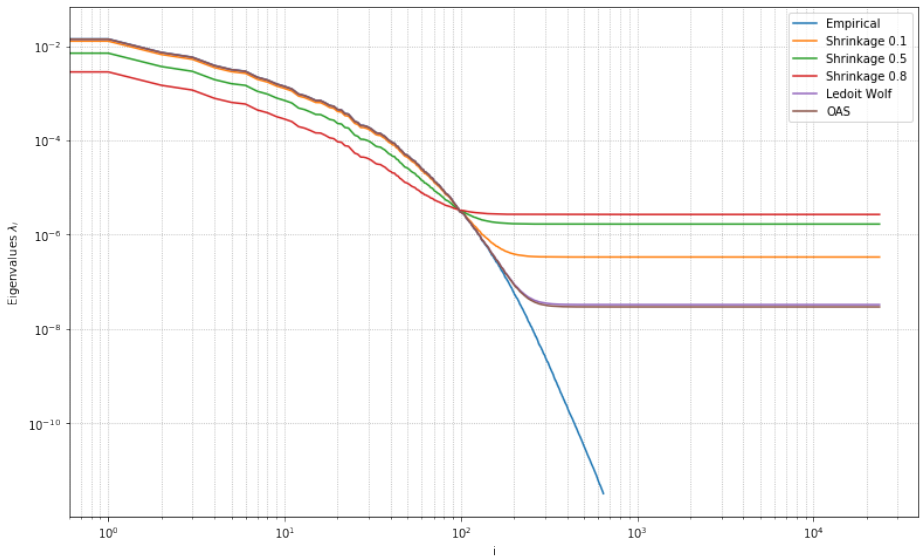


**Figure 4.2:** Eigenvalues Covariance Comparison - Tracer

As we can see the shrinkage algorithm allows to have a positive definite matrix that is very close to the original covariance matrix. It shares a lot of eigenvalues with the

sample covariance matrix. This varies in function of the shrinkage constant $\rho$. For arbitrary values, $\rho = 0.1$, $\rho = 0.5$ and $\rho = 0.8$, we see that the smaller the shrinkage constant is, the closer we get to the sample covariance. For large constant value, we see that the upper part of the spectrum deteriorates completely, meaning that we don't have a good estimation anymore. This is perfectly logical with regards to the definition of the shrinkage which is a convex linear combination between a diagonal matrix and the sample covariance.

For LW and OAS, the shrinkage variable computed is close to $0.1$. The spectrums are almost identical and are resulting in very similar covariances matrixes. We see that the upper part of the spectrum is almost identical to the sample covariance matrix, but that the lower part is truncated.

Globally, we see that with shrinkage the small eigenvalues get larger and that the large ones get smaller. We observe also that the determinant becomes positive, and also that it leads to a positive definite matrix.

As OAS gives us theoretically an optimal $\rho$ with very small error, therefore, we decide to use the **OAS estimation** of the covariance matrix for the optimisation procedure to follow.

## 4.2 Comparison Optimisation Algorithms

We have defined the three algorithms allowing for a near-optimal placement of the sensors. We have also defined an approximation of the GPs. To compare their performance and measure their limitation, we have run them on a small dataset, a subset of our main one.

### 4.2.1 Conditions of the Experiment

We have defined a sphere of radius $25$m, centred close to the centre, in the middle of the propagation beam, at position $[60, 35, 0]$ in which are contained $3'130$ points of the mesh. By taking the intersection between those points and our selection defined in 3.3, we can reduce the number of points $|S|$ to $1'295$. The candidates are presented in Figure 4.3.

We compute the OAS covariance for this dataset and we will use in in the rest of the experience. The eigenvalues decomposition is shown for both the empirical covariance and the OAS estimate in Figure 4.4. As shown previously we have a spectrum that is close to the original sample covariance, only with a truncated lower part.

For the *Tracer* we have a sample covariance that is **negative definite** with *logDet* of $-39'755.06$. The OAS approximation allows us to get a **positive definite** covariance matrix with *logDet* of $-19'447.95$.

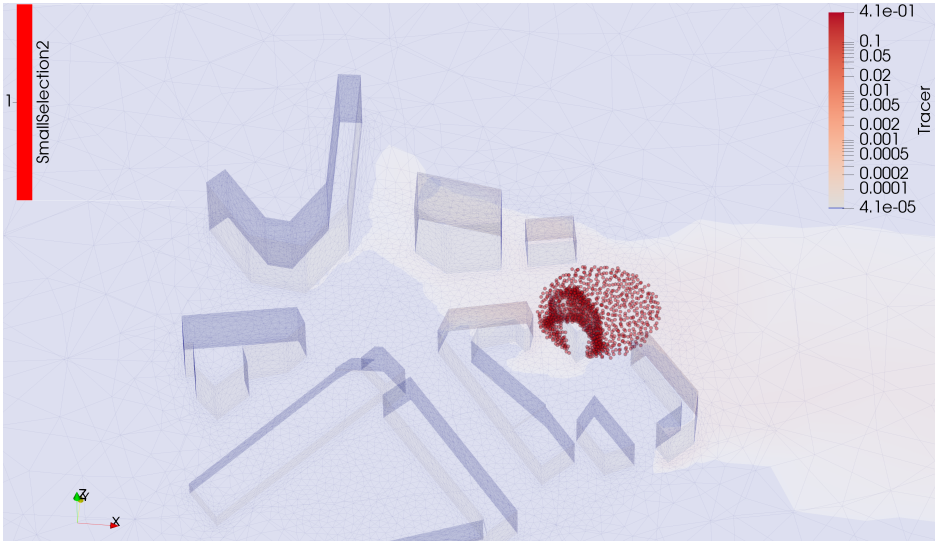


**Figure 4.3:** Small Subset Candidates

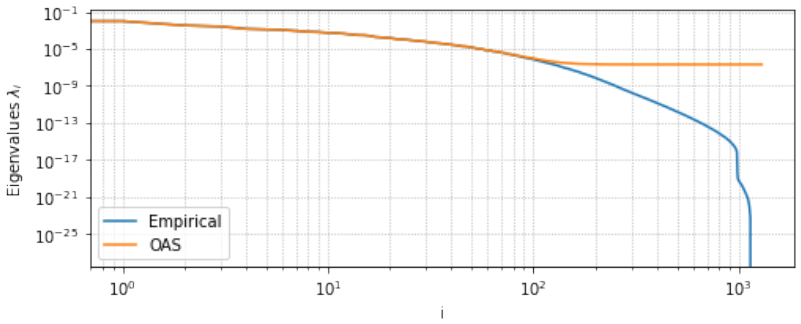


**Figure 4.4:** Spectrum of Empirical and OAS covariances for Tracer

Next, we are going to optimise in this space $k = 10$ points with the different algorithms and compare them based on the computation time and the distance between the datasets using the NN distance defined in 3.12.

### 4.2.2 Greedy and Lazy Optimisation

First, we optimise using the two first algorithms, the **greedy** and the **lazy** versions of the near-optimal sensor positioning algorithms. We will consider the output of the first algorithm as the reference for this comparison. This is because the first algorithm is the most reliable version as it covers every candidate point in its loops and relies on full GP implementation.

We observe immediately that output sets given by the two methods are the same. The main difference is as expected the computation time which is much larger in the first algorithm. The Greedy Optimisation gives a result after $1043.27$s and the *lazy* Optimisation gives results after $153.51$s. The second algorithm is, for this small dataset, seven times faster than the greedy one: this is a significative difference.

The location of those points optimal points $\mathcal{A}^*$ is shown in green in the Figure 4.5 and table 4.2. We observe that the points are well spread across space and along the wind direction. As the wind is propagating the tracer concentration, it makes sense that the sensors must be placed along this direction.

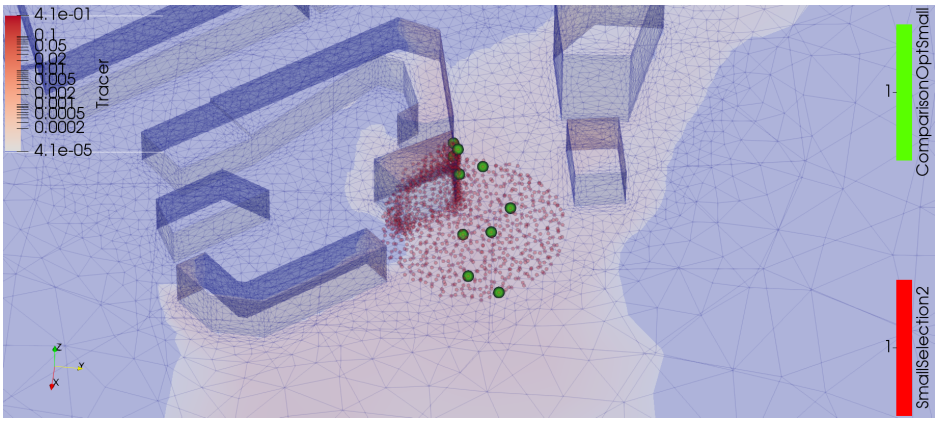


**Figure 4.5:** Optimal Points Location: Illustation

| $\mathcal{A}^*$ | 52731 | 47876 | 3078 | 19782 | 26045 | 30511 | 26754 | 81507 | 11608 | 3903 |
|---|---|---|---|---|---|---|---|---|---|---|
| X | 61.31 | 43.55 | 77.31 | 38.75 | 48.23 | 50.48 | 52.36 | 43.10 | 82.40 | 62.01 |
| Y | 40.06 | 27.62 | 35.55 | 35.26 | 29.38 | 30.04 | 44.37 | 27.52 | 44.48 | 32.39 |
| Z | 1.52 | 16.40 | 1.45 | 1.80 | 19.53 | 11.79 | 1.62 | 10.23 | 1.96 | 0.20 |

**Table 4.2:** Optimal Points Locations

We also can plot the Mutual Information Gain $\delta_{y^*}$ at each sensor placement on Figure 4.6. What we can see confirms that our Mutual Information is sub-modular, as the Mutual Information Gain is monotonically decreasing.

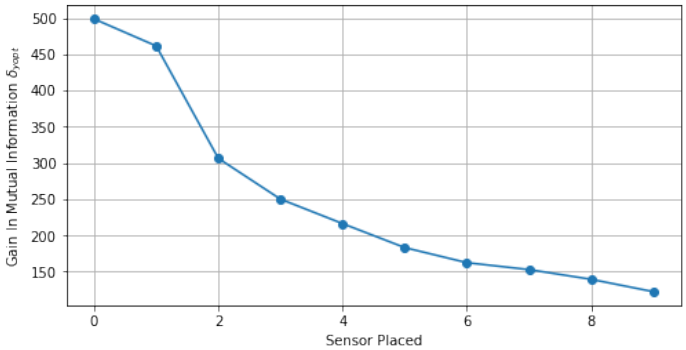


**Figure 4.6:** Mutual Information Gain for each sensor - Greedy & Lazy Algorithms

### 4.2.3 Local Kernels Optimisation

The 3rd Algorithm that we are going to use is the one that is scalable to larger datasets. This scalability is dependent on the parameter $\epsilon$, the thresholding parameter of the covariance. As a consequence the number of selected covariates $|N(y_{opt}, \epsilon)| \leq d$ is fluctuating and influences greatly the speed of the algorithm. It is directly linked to the GP and the size of the covariance matrix to invert.

This is why the threshold $\epsilon$ has to be chosen carefully. We are going to show that the speed, the NN distance and the parameter d varies according to $\epsilon$, before choosing a value that will be used for the full-scale optimisation. We compute of a range of values for $\epsilon$ all those results and show them in table 4.3 and in Figure 4.7.

| Threshold $\epsilon$ | $10^{-10}$ | $10^{-9}$ | $10^{-8}$ | $10^{-7}$ | $10^{-6}$ | $10^{-5}$ | $10^{-4}$ | $10^{-3}$ | $10^{-2}$ | $10^{-1}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| Distance [m] | 0.00 | 0.00 | 0.00 | 0.00 | 1.248 | 2.783 | 4.693 | 11.058 | 11.058 | 11.058 |
| Time [s] | 1660.85 | 1617.92 | 1500.63 | 1151.68 | 520.36 | 60.88 | 2.44 | 1.74 | 3.25 | 2.13 |
| Average d | 1289.40 | 1288.20 | 1284.80 | 1245.60 | 1029.60 | 510.50 | 15.20 | 0.00 | 0.00 | 0.00 |

**Table 4.3:** Local Kernel Results - Tracer

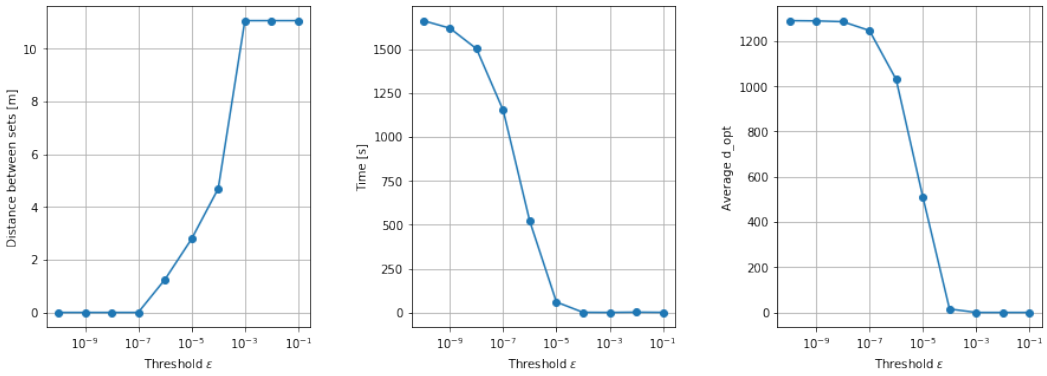


**Figure 4.7:** Local Kernels: Distance, Time and d, in function of $\epsilon$ - Tracer

We see that computation time is correlated with the average number of correlates d. For a threshold bellow $10^{-7}$, we see that the algorithm selects almost every point available and therefore we have computation times of the order of the greedy algorithm. The optimised set of points is then also the same as the greedy and lazy algorithms.

By increasing the threshold, we find that the computation time and the number of covariates decrease at the expense of the accuracy represented by the increase in the NN distance between this dataset and the optimal one. We observe also that for $\epsilon > 10^{-3}$, the number of covariates selected d is equal to zero which means that the algorithm updates every position at each iteration and uses for the optimisation only the values of the original covariance, without conditioning them, as the observed set

is empty.

Therefore we need to **fix the threshold** at around $\epsilon_{opt} = 10^{-6}$, so that the error is still small and the computation time substantially reduced. This threshold will be used on the full-scale optimisation of Tracer Concentration Data.

By relocating the centre of the selection sphere, we observe that the optimal threshold is very different from one zone to the other. For example, if we place the subset in a zone where the tracer data is almost always null, we see that the threshold needs to be much smaller to get any results. This is why the value we have chosen was optimised for an area where there is some relevant data.

### 4.2.4 Approximated Gaussian Processes

Alternatively, we use the approximation method developed earlier relying on the TSVD of the data, applied to algorithm 2. We proceed to the optimisation of the dataset using different values for the truncation parameter $\tau$. We then plot the computation time and the set distance to the results of the greedy algorithm in function of the truncation parameter in Figure 4.8.

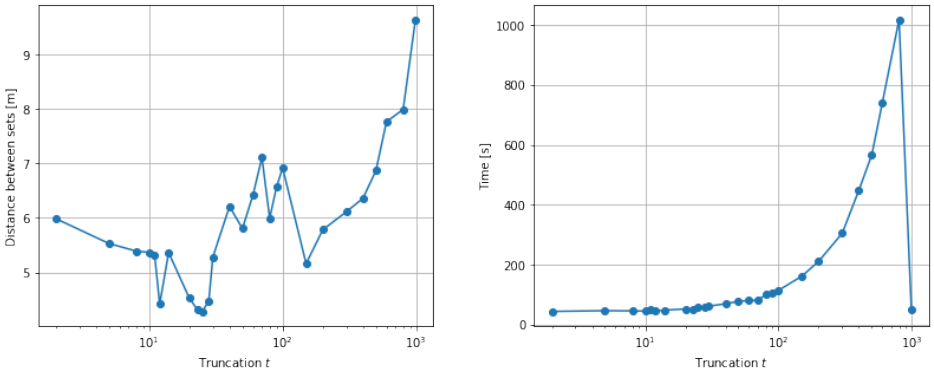


**Figure 4.8:** TSVD Approximation: Distance and Time in function of $\tau$

As we can see the computation time increases exponentially with $\tau$ and it is then reduced when $\tau$ gets larger than $n = 988$. For the NN distance between those sets and the optimal sets of algorithm 1, we see that it is quite low at the for a small $\tau$, and it reaches a minimum at $\tau_{opt} = 25$. For larger values, the distance is strongly increasing. We will keep this method and the optimal truncation parameter, for testing on the full dataset in the following section.

It is important to mention that the result is here empirical, and that some methods providing optimal parameters are given by Arcucci et al. (2019). In this publication, the dataset used is similar and the optimal truncation found is of the same order.

## 4.3 Full Scale Optimisation Results

After the analysis of different algorithm results on a smaller dataset, we are now going to use the most scalable algorithms with the appropriate parameters on the full dataset. The set of points used is the preselected dataset containing $23'643$ locations.

In this section, we will present the results of two optimisations, compare them in term of computation speed and proximity.

### 4.3.1 Local Kernel Algorithm

We first use the Local Kernel Algorithm 3. The covariance between the points is computed using the OAS estimator.

For this experiment we used the previously defined value of threshold: $\epsilon = 10^{-6}$. Such that the number of points highly correlated to the last positioned sensor is $|N(y_{opt}, \epsilon)| \leq d$.

The algorithm computes the following results in $23797.2$ seconds or $6.61$ hours.

We are computing for each iteration the size of this set. This number is directly linked to the size of the covariance matrix being inverted in the GPs. As we are placing 10 sensors, we have 10 values of this quantity that is computed at each iteration of the main loop. Results can be found in table 4.4. The average value is $4'374.7$. This shows that indeed the optimisation problem is solved in a reduced time as it doesn't use all the $23'643$ locations of our dataset, but $18.50\%$ of it.

| Sensor | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $\lvert N(y_{opt}, \epsilon)\rvert$ | 4338 | 4938 | 5066 | 5362 | 4239 | 4164 | 3377 | 4632 | 3511 | 4120 |

**Table 4.4:** Number of highly correlated points at each iteration

We visualise the position of the optimal set of sensor points $\mathcal{A}^*$ at different scales in Figure 4.9. The coordinates of the points are expressed in the table 4.5.

| $\mathcal{A}^*$ | 56588 | 52731 | 73959 | 43278 | 3078 | 19782 | 56257 | 55640 | 10357 | 54786 |
|---|---|---|---|---|---|---|---|---|---|---|
| X | 41.61 | 61.31 | 30.32 | 22.79 | 77.31 | 38.75 | 91.19 | 50.10 | 62.13 | 1.62 |
| Y | 27.21 | 40.06 | 26.06 | 25.81 | 35.55 | 35.26 | 35.16 | 29.00 | 45.23 | 19.99 |
| Z | 16.73 | 1.52 | 11.19 | 11.79 | 1.45 | 1.80 | 1.82 | 16.41 | 1.64 | 13.81 |

**Table 4.5:** Optimal Points Locations - Tracer

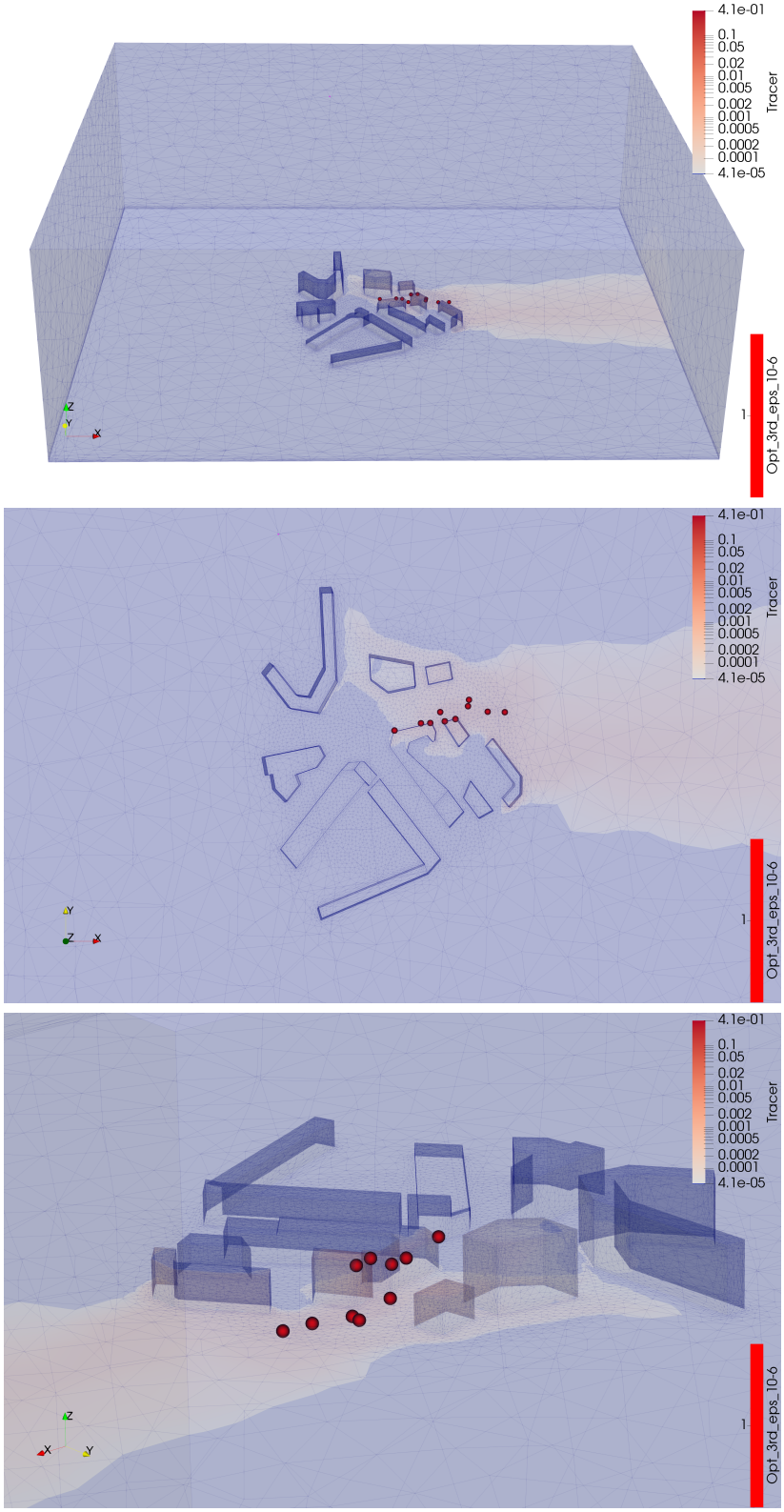


**Figure 4.9:** Position of the Optimal Set

**Description**

The location of our optimal set of points is split between points situated on the buildings (5 points) and around the ground level (5 points). They are situated in the middle of the tracer concentration beam and are close to the centre of the domain where the mesh density is maximal.

The points are roughly aligned in the wind direction, which is a good thing because the tracer is propagating itself from the centre of the domain, pushed by the wind in the east direction. Points along this direction are less correlated with each other than points perpendicular to this direction, as the tracer reaches them at the same time. Therefore it makes sense that the selected points are roughly along the wind direction as it will maximise the information retrieved during the propagation of the tracer.

Finally we observe the Mutual information gain $\delta_{y^*}$ for each placed sensor on Figure 4.10. It confirms that our MI is sub-modular as here again the decreasing of the MI difference is monotonic.

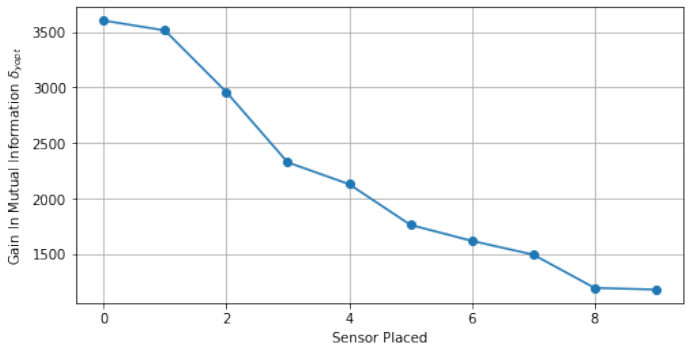


**Figure 4.10:** Mutual Information Gain for each sensor - Algorithm 3

### 4.3.2 Lazy Algorithm with TSVD

We apply on the same dataset the 2nd Algorithm with the approximated GP that we defined earlier. We use the truncation parameter $\tau_{opt} = 25$. The results we obtain are displayed in table 4.6 and Figure 4.11, along with the previous optimal results.

This algorithm computes the following results in $18083.87$ seconds or $5.02$ hours.

**Description**

Here again, the location of our optimal set of points is split between points situated on the buildings (3 points) and around the ground level (7 points). The locations are more spread than the other set of points. Two locations are even at the border of the candidate dataset. The NN distance between this set and the previous one is

13.84m, which is not very important with regards to the size of the space.

| $\mathcal{A}^*$ | 38726 | 14276 | 91348 | 5338 | 40994 | 29626 | 65851 | 65734 | 851 | 2293 |
|---|---|---|---|---|---|---|---|---|---|---|
| X | -50.50 | 35.63 | 43.05 | 137.58 | 80.34 | 27.79 | 49.54 | 54.59 | 77.54 | 62.03 |
| Y | 48.85 | 58.69 | 27.51 | 55.78 | 76.20 | 26.24 | 28.88 | 25.42 | 34.82 | 41.92 |
| Z | 14.84 | 0.20 | 7.76 | 0.20 | 0.20 | 12.06 | 17.65 | 3.80 | 0.20 | 0.20 |

**Table 4.6:** Optimal Points Locations TSVD - Tracer

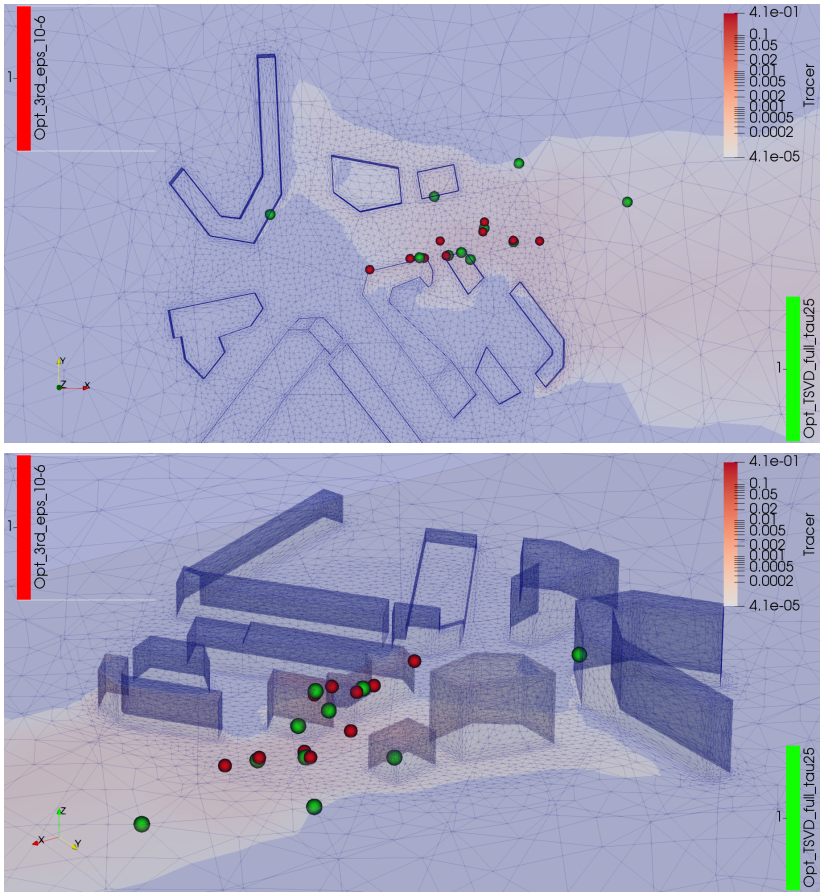


**Figure 4.11:** Position of the Optimal Set: Main View $\tau = 25$

Finally we observe the Mutual information gain $\delta_{y^*}$ for each placed sensor on Figure 4.12 is also constantly decreasing. However we can observe that it is not linear, as it was in the small set experimentation and in the Algorithm 3 optimisation.

## 4.4 Application to Variational DA

As we have seen, one way of applying the results to a real situation is to proceed to a Variational Data Assimilation of the points selected. By taking results of a simulation

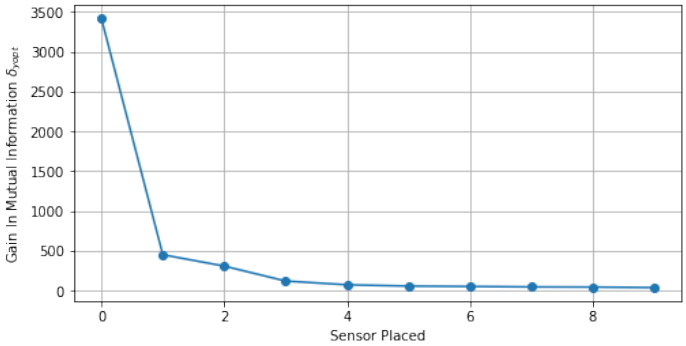


**Figure 4.12:** Mutual Information difference for each sensor - TSVD

and applying DA on our selected points we can try to see if the results are usable in the context of the MAGIC project.

### 4.4.1 Parameters of the Experiment

For this experiment, we chose to consider the same simulation data than previously: the 3D smallLSBU. We choose as the observation state, the step $988$ of the simulation, and as background state, the step $100$ of the simulation.

We compute the Variational DA on the 10 optimal points selected by the algorithms and compute the background error and the error after the DA procedure.

### 4.4.2 VarDA on Local Kernel Results

First, we run the VarDa Algorithm and obtain the following.

| MSE [xDA] | MSE [xB] | Imp. [xDA / xB] |
|---|---|---|
| $7.7375 \cdot 10^{-17}$ | $0.9923$ | $7.7975 \cdot 10^{-17}$ |

**Table 4.7:** VarDA on Local Kernel Results.

#### Comparison to Random

To compare those results to another set of points, we proceed as follows. Around every single point of $\mathcal{A}^*$, we select randomly (uniformly) a point located within a radius of $R = 10m$. This allows us to compare the results to some random set that is close to the optimum. We then proceed to $1'000$ different random selection and compute the average errors and their standard deviations, stated in table 4.8.

| | Average | Std Dev |
|---|---|---|
| MSE [xDA] | $6.1008 \cdot 10^{-17}$ | $4.1680 \cdot 10^{-17}$ |
| MSE [xB] | $0.6538$ | $0.1718$ |
| Imp. Ratio [xDA / xB] | $9.3313 \cdot 10^{-17}$ | - |

**Table 4.8:** VarDA on $1'000$ Near Random Points: Local Kernel Results

**Analysis**

We can see that the error of the optimal set $\mathcal{A}^*$ is very small on the DA results, almost negligible. The improvement form the pre-assimilation background error is of almost $100\%$. This shows that this set of points has some good properties for Data Assimilation.

By comparing those results to the $1'000$ random samples, we see that the average is very close to the error of the optimal set. However, the improvement from the background error is smaller for the random sets. In this way, we have better results for the optimal set of points.

### 4.4.3 VarDA on Gaussian Approximation Results

We then run the same algorithm on the optimal set found with the lazy algorithm combined with TSVD GP approximation.

| MSE [xDA] | MSE [xB] | Imp. Ratio [xDA / xB] |
|---|---|---|
| $9.8945 \cdot 10^{-17}$ | $1.1354$ | $8.7145 \cdot 10^{-17}$ |

**Table 4.9:** VarDA on TSVD Results.

**Comparison to Random**

To compare those results to another set of points we proceed as previously. The results for $1'000$ random sets sampled at a $10$m radius from each optimal points are given in table 4.10.

| | Average | Std Dev |
|---|---|---|
| MSE [xDA] | $6.6843 \cdot 10^{-17}$ | $4.9458 \cdot 10^{-17}$ |
| MSE [xB] | $0.6725$ | $0.2148$ |
| Improvement Ratio [xDA / xB] | $9.9381 \cdot 10^{-17}$ | - |

**Table 4.10:** VarDA on 500 Random Points: TSVD Results

**Analysis**

We observe the same phenomenon than with the *Tracer* Local Kernel results. The DA error is negligible and the Improvement is better with our optimal points.

### 4.4.4 Limitations of the Application

Even if the error magnitude is not decreasing, we can see an improvement in the value of the computed ratio between the DA error and the background error. It has been shown by Arcucci et al. (2019), that when the simulation is continued after DA, the an initial improvement is propagated and an initial better improvement leads to better results overall for both the sensors and un-monitored spaces.
We are here measuring the error directly after the assimilation and only on the 10 assimilated points. As VarDA is an optimisation problem that minimises this error, considering the small number of points and the fact that there is no dimensionality reduction of the deviation matrix, it is quite natural that this error is negligible.

To draw more conclusions on the validity of our set of points, we would need to continue the simulation after the DA procedure and compare after some time the states of all the points of the space with different assimilated sets of points.

# Chapter 5

# Conclusion and Future Works

The objectives of the project were to explore, implement and analyse the near-optimal sensor-positioning in an environmental context. We used data from an air-flow simulation of pollution in a London neighbourhood as a basis for our study. We were able to extract and analyse several states present in the simulation. We have implemented the algorithms proposed by Krause et al. (2008). We have discovered those limitations and have overcome them.

At first, we have explored the ways of reducing the size of the dataset. We have selected locations based on the value of the pollution tracer and the shape the physical space. We have analysed the ways of modelling the data available by estimating its covariance matrix. We have found that the sample covariance was not a good estimator, and used the shrinkage method to have a valid covariance matrix. Finally, we have implemented methods to reduce the computational cost of the Gaussian Process based optimisation. An approach relying on the usage of only highly correlated locations, and another approach using dimensionality-reduction techniques.

In order to compare all the methods available, we have tested them on a small subset of the data and verified their consistency. After selecting the optimal parameters for the algorithms, we have applied the scalable ones to the full dataset and obtained a set of optimal points.

As an application, we have finally used those optimal points with a Data Assimilation algorithm. We have found out that those were optimally assimilated to the simulation.

We could think of several improvements and further developments that could be undertaken in this project.

The estimation of a better covariance matrix for modelling the data is certainly a good approach. There exist alternatives to the methods used in the project that could lead to more accurate data modelling.

The application to the DA could be also further improved by continuing the simula-

tion after the assimilation step. We would be able to compare the error on the whole dataset between post-DA simulation with different sets of DA points.

Finally, one other idea for handling the near-optimal sensor optimisation would be to take a different approach and leave aside the methods based on Mutual Information criterion. We could keep the idea of using GPs to model the data and keep for example the *inducing points* defined in sparse GP methods as the optimal set of sensors. As it is a crucial subject for the research, there is a lot more to improve and discover in the field.

# Appendix A

# Ethical and Professional Considerations

Throughout this project, I wasn't confronted by any ethical questions and have therefore not checked any item of the ethical consideration list. My project explored the applications of a study proposing an algorithm for near-optimal sensor positioning. Optimising sensor positions is a very broad subject that has applications in most technical fields. I have applied those algorithms in an environmental context. By placing a reduced number of points optimally, we can help to improve the results of the simulations on pollution concentrations. We can also consider that such application can help to increase the number of similar studies by reducing the cost of actual sensor deployment. Our project can, therefore, be considered as respecting the ethical and professional considerations requirements.

# Appendix B

# Description of the Code

The codes created for this project are available in the **GitHub** repository of the project at the following address: `https://github.com/adrianlwn/MScProject_OptSensors_GP`.

We have developped some jupyter notebooks in list also the *Jupyter Notebooks* and python scripts used to run and produce all the results of our project:

- `parameters.py`: File containing the parameters such as the time interval times to import, the crop of the vtk files, the shape of the buildings.
- `Sensor Optimisation 3rd Algorithm.ipynb`: Optimisation Full Scale with 3rd Algorithm.
- `Sensor Optimisation Comparison 1-2-3.ipynb`: Optimisation Small Scale with All Algorithms.
- `Sensor Optimisation TSVD.ipynb`: Optimisation Full Scale with 2nd Algorithm and TSVD GP Approximation.
- `Covariance Comparison Plots.ipynb`: Computation of all the covariances approximations.
- `Data Assimilation Randomized Points.ipynb`: Computation of the DA Application with optimal points and with randomized points.
- `Preprocessing.ipynb`: Preprocessing Steps for the Data Analysis.
- `Data Visulalisation.ipynb`: Some Early Visualisation of the Data.

The fluidity library is stored outside of the code in the folder `../data/fluidity-master/`.
The VTK files are store in the folder `../data/small3DLSBU/`.
The temp files and various result vtk are stored in the folder `../data/data_temp/`.

The main functions developed during this project, especially the optimisation algorithms, are placed in a python library named `utils` containing the following files:

- `data_handling.py`: All the functions used to handle the vtk files, extract the subsets for the optimisation. It also allows to save the results in the VTK files for the purpose of visualisation.
- `sensor_optimisation.py`: All the functions needed to run the standard version of the Optimisation algorithms. Those functions take the covariance matrix in parameter.
- `sensor_optimisation_tsvd.py`: All the functions needed to run the modified versions of the Optimisation algorithms. Those functions take the data in parameter.
- `config.py`: This script containts some usefull paths to find the data and save the temporary files and results.
- `data_assimilation.py`: This file contains functions to realise the data assimilation.

# Bibliography


Arcucci, R., Mottet, L., Pain, C., and Guo, Y.-K. (2019). Optimal reduced space for Variational Data Assimilation. *Journal of Computational Physics*, 379:51–69. pages 19, 30, 37, 44

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 4

Caselton, W. and Zidek, J. (1984). Optimal monitoring network designs. *Statistics & Probability Letters*, 2(4):223–227. pages 13

Chen, Y., Wiesel, A., Eldar, Y. C., and Hero, A. O. (2010). Shrinkage Algorithms for MMSE Covariance Estimation. *IEEE Transactions on Signal Processing*, 58(10):5016–5029. pages 9, 10

Cover, T. M. and Thomas, J. A. (1991). *Elements of information theory*. Wiley series in telecommunications. Wiley, New York. pages 12, 13

Cressie, N. A. C. (1993). *Statistics for spatial data*. Wiley series in probability and mathematical statistics. Wiley, New York. pages 4

Deisenroth, M. P., Faisal, A., and Ong, C. S. (2018). Mathematics for Machine Learning. page 43. pages 5

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 9

Foster, L., Waagen, A., Aijaz, N., Hurley, M., Luis, A., Rinsky, J., Satyavolu, C., and Com, M. (2009). Stable and Efficient Gaussian Process Calculations. page 26. pages 6

Hansen, P. C. (1987). The truncatedSVD as a method for regularization. *BIT*, 27(4):534–553. pages 26

Krause, A., Singh, A., and Guestrin, C. (2008). Near-Optimal Sensor Placements in Gaussian Processes: Theory, Efficient Algorithms and Empirical Studies. page 50. pages 1, 6, 8, 11, 12, 13, 14, 15, 17, 27, 28, 45

Ledoit, O. and Wolf, M. (2003a). Honey, I Shrunk the Sample Covariance Matrix. page 22. pages 9

Ledoit, O. and Wolf, M. (2003b). Improved estimation of the covariance matrix of stock returns with an application to portfolio selection. *Journal of Empirical Finance*, 10(5):603–621. pages 9

Ledoit, O. and Wolf, M. (2004). A well-conditioned estimator for large-dimensional covariance matrices. *Journal of Multivariate Analysis*, 88(2):365–411. pages 10

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 5

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 14

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 8

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 4, 5, 8, 11

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 3

Titsias, M. K. (2009). Variational Learning of Inducing Variables in Sparse Gaussian Processes. page 8. pages 7

Wilson, A. G. and Nickisch, H. (2015). Kernel Interpolation for Scalable Structured Gaussian Processes (KISS-GP). page 10. pages 7