Imperial College
London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Gaussian Processes for Optimal Sensor Position

MSc Project Report

Author:
Adrian LÖWENSTEIN

Supervisor:
Rossella ARCUCCI
Miguel MOLINA-SOLANA

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

August 20, 2019

Contents

1 Introduction **2**
- 1.1 Introduction . . . 2
- 1.2 Problem Statement . . . 2
- 1.3 Aim of the project . . . 2

2 Literature Review **3**
- 2.1 The MAGIC Project . . . 3
- 2.2 Gaussian Processes . . . 4
  - 2.2.1 Multivariate Gaussian Distribution . . . 4
  - 2.2.2 Prediction with Gaussian Processes . . . 4
  - 2.2.3 Scalable Gaussian Processes . . . 5
- 2.3 Covariance Matrix . . . 8
  - 2.3.1 Properties of Covariance . . . 8
  - 2.3.2 Covariance Estimation . . . 8
  - 2.3.3 Kernel Functions . . . 10
- 2.4 Sensor Position Optimisation . . . 11
  - 2.4.1 Placement Criterion . . . 11
  - 2.4.2 Approximation Algorithm . . . 13
  - 2.4.3 Improvements over the Algorithm . . . 14
- 2.5 Variational Data Assimilation . . . 16

3 Implementation **17**
- 3.1 Practical Informations . . . 17
  - 3.1.1 Availability of the code . . . 17
  - 3.1.2 Environment . . . 17
  - 3.1.3 Experimental Conditions . . . 18
- 3.2 Data Analysis . . . 18
- 3.3 Preselection of the Data . . . 19
  - 3.3.1 Selection of a working subset of the data . . . 20
  - 3.3.2 Selection of a subset at human level . . . 21
  - 3.3.3 Combined Selection . . . 23
- 3.4 Implementation of Covariance Matrix . . . 23
- 3.5 Implementation of Gaussian Processes . . . 24
  - 3.5.1 Classical GPs . . . 24
  - 3.5.2 GPs Approximation with TSVD . . . 24
  - 3.5.3 Other Approaches . . . 25

3.6 Implementation of the Optimisation 26
3.6.1 26
3.6.2 Comparison tools : Distance Between Sets 27
3.7 Implementation of VarDA 28

4 Results and Analysis **29**
4.1 Covariance 29
4.1.1 Sample Covariance 29
4.1.2 Shrinkage Covariance 30
4.1.3 Arbitrary Covariance Function 30
4.2 Comparison Optimisation Algorithms 30
4.2.1 Conditions of the Experiment 31
4.2.2 Greedy and Lazy Optimisation 32
4.2.3 Local Kernels Optimisation 33
4.2.4 Approximated Gaussian Processes 34
4.2.5 Arbitrary Covariance 34
4.3 Optimisation Results 35
4.3.1 Local Kernel Algorithm 35
4.3.2 Lazy Algorithm with TSVD 37
4.4 Validation with VarDA 38

5 Conclusion **39**

A Description of the Code **40**

Todo list

Write Intoduction 2
Write Problem Statement : Why we need that ? What solution ? 2
Explain the steps of the project r eport 2
Mention the main assumption that the data needs to be gaussianly distributed and iid samples (Maybe already explained) 4
Speak of intuition of covariance in function of the distance from center 10
Explain Data Assimilation 16
Make sure that the packages are all listed 17
Important to explain how the gaussian processes were implemented. Challenge of the inversion. Classical approach and TSVD approach and show simple computation times and error results. 24
Results add discussion on the cost of TSDV refer to the email 25
check the last paragraph and maybe move it 26
Explain every algorithm implemented, their difference. This is important for the result section where sample results for those algorithm will be shown later on 26
Write this in the end only if the validation works well 28
Explain why only on full set and not on comparison set 28
Comment and Compare shrinkage versions 30
Maybe not 30
Add in literature review // Implementation the Kernels definitions 34
Add Validation Results 38
Change Conclusion of the Project 39

Chapter 1

Introduction

1.1 Introduction

Write Intoduction

1.2 Problem Statement

Write Problem Statement : Why we need that ? What solution ?

1.3 Aim of the project

Structure of the Project

Explain the steps of the project r eport

Project Proposal:

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

Dataset:

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

Chapter 2

Literature Review

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

2.1 The MAGIC Project

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

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

As explained by Song et al. (2018), the test site which has been selected to conduct the study is a real urban area located in London South Bank University (LSBU) in Elephant and Castle, London. In order to investigate the effect of ventilation on the cities problem, researchers in the MAGIC project have created simulations and experiments both in outdoor and indoor conditions, on the test site. They used wind

tunnel experiments and computational fluid dynamics (CFD) to simulate the outdoor environment. Further works include the development of reduced-order modelling (ROM) in order to make faster the simulations while keeping a high level of accuracy (Arcucci et al., 2018).

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

2.2 Gaussian Processes

In this 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 (**?**). Applied in particular in Geostatistics with methods known as *kringing*, and in Meteorology. Gradually GP started to be used in more general cases for regression. Nowadays it is often used in the context of Machine Learning.

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

Mention the main assumption that the data needs to be gaussianly distributed and iid samples (Maybe already explained)

2.2.1 Multivariate Gaussian Distribution

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

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

2.2.2 Prediction with Gaussian Processes

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

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

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

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

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

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

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

2.2.3 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 $m << p$.

Sparse Kernel

The **sparse kernel** approach is the idea to reduce the importance of the points far from each other and imposing sparsity of the covariance by setting to zero the ele-

ments 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.

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 idea.

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. In order to further improve this method the idea of *Partially Independent Training Conditional* (PITC) was introduced. Here we consider that not all the training points are independent, but only some of them. This leads to an independence by blocks, and an block-diagonal covariance approximation : $\tilde{\boldsymbol{Q}}_{nn} = blkdiag(\boldsymbol{K}_{nn} - \boldsymbol{Q}_{nn})$.
In those methods, the choice of the m *inducing points* is crucial to enable a good approximation.

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

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

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

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

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

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

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

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 the best inducing points representing the observations of the original GP. Once this step is done, they allow a very computationaly efficient prediction. 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. This creates a new bottleneck.

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 estimate of the covariance matrix between the points of our mesh.

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

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

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 $\boldsymbol{\Sigma}$. 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).

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 for the the purpose of improving 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 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) :

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

The shrinkage oracle estimator is defined such as :

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

The optimisation problem was expressed as follows by Ledoit and Wolf (2004), ($\|\cdot\|_F$ being the Frobenius matrix norm) :

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

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

$$\tag{2.14}$$

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 dependant of 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.

2.3.2.3 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 $0 < \rho^*_{OAS} \leq 1$:

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

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

2.3.3 Kernel Functions

Role of parametric kernel function in gaussian processes. learning the parameter on real data and predicting in between. Smoothness of the prediction,Likelyhood. learned at one specific time sample, loosing information on eht propagation ! Covariance .

No chosen because krause said it was bad.

Speak of intuition of covariance in function of the distance from center

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

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

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

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 in an optimal way is called in spatial statistics, *sampling* or *experimental design*. We want to find the optimal way to place a number of k sensors (indexed by $\mathcal{A}$) inside the set of possible sensor locations $\mathcal{S}$. So that we have $\mathcal{A} \subseteq \mathcal{S} \subseteq \mathcal{V}$.

For the rest of this section, we assume that we have at our disposal a good estimate of the covariance matrix between each point. In practice this is not that obvious as we have seen in section 2.3. The following is valid for any covariance matrix that is symmetric and positive-definite.

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

2.4.1 Placement Criterion

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

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

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) :

$$\begin{aligned} H(\mathcal{X}_{\mathcal{V}\setminus\mathcal{A}}|\ \mathcal{X}_\mathcal{A}) &= H(\mathcal{X}_{\mathcal{V}\setminus\mathcal{A}},\ \mathcal{X}_\mathcal{A}) - H(\mathcal{X}_\mathcal{A}) && (2.25)\\ &= H(\mathcal{X}_\mathcal{V}) - H(\mathcal{X}_\mathcal{A}) && (2.26)\end{aligned}$$

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

$$\begin{aligned} \mathcal{A}^* &= \arg\min_{\mathcal{A}\subseteq\mathcal{V}:|\mathcal{A}|=k} H(\mathcal{X}_{\mathcal{V}\setminus\mathcal{A}}|\ \mathcal{X}_\mathcal{A}) && (2.27)\\ &= \arg\min_{\mathcal{A}\subseteq\mathcal{V}:|\mathcal{A}|=k} H(\mathcal{X}_\mathcal{V}) - H(\mathcal{X}_\mathcal{A}) && (2.28)\\ &= \arg\max_{\mathcal{A}\subseteq\mathcal{V}:|\mathcal{A}|=k} H(\mathcal{X}_\mathcal{A}) && (2.29)\end{aligned}$$

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 $\mathcal{V}\setminus\mathcal{A}$. Using the definitions of MI provided by Cover and Thomas (1991, p. 19) :

$$\begin{aligned} \mathcal{A}^* &= \arg\max_{\mathcal{A}\subseteq\mathcal{V}:|\mathcal{A}|=k} I(\mathcal{X}_{\mathcal{V}\setminus\mathcal{A}},\ \mathcal{X}_\mathcal{A}) && (2.30)\\ &= \arg\max_{\mathcal{A}\subseteq\mathcal{V}:|\mathcal{A}|=k} H(\mathcal{X}_{\mathcal{V}\setminus\mathcal{A}}) - H(\mathcal{X}_{\mathcal{V}\setminus\mathcal{A}}|\mathcal{X}_\mathcal{A}) && (2.31)\end{aligned}$$

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 is able to approximate in polynomial time the optimal solution, with a constant factor guarantee.

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

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

$$y* = \arg\max_{y \in \mathcal{S}\backslash A} MI(\mathcal{A} \cup y, \bar{\mathcal{A}}) - MI(\mathcal{A}, \mathcal{V}\backslash\mathcal{A}) \tag{2.32}$$

$$= \arg\max_{y \in \mathcal{S}\backslash A} H(y|\mathcal{A}) - H(y|\bar{\mathcal{A}}) \tag{2.33}$$

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

$$y* = \arg\max_{y \in \mathcal{S}\backslash A} \frac{1}{2}\log K_{y|\mathcal{A}} - \frac{1}{2}\log K_{y|\bar{\mathcal{A}}} \tag{2.34}$$

$$= \arg\max_{y \in \mathcal{S}\backslash A} \log\left(\frac{K_{y|\mathcal{A}}}{K_{y|\bar{\mathcal{A}}}}\right) \tag{2.35}$$

$$= \arg\max_{y \in \mathcal{S}\backslash A} \frac{K_{y|\mathcal{A}}}{K_{y|\bar{\mathcal{A}}}} \tag{2.36}$$

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

$$y* = \arg\max_{y \in \mathcal{S}\backslash A} \frac{K_{yy} - K_{y\mathcal{A}}K_{\mathcal{A}\mathcal{A}}^{-1}K_{\mathcal{A}y}}{K_{yy} - K_{y\bar{\mathcal{A}}}K_{\bar{\mathcal{A}}\bar{\mathcal{A}}}^{-1}K_{\bar{\mathcal{A}}y}} \tag{2.37}$$

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

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 :

$$MI(\hat{\mathcal{A}}) \geq (1 - \frac{1}{e}) \cdot MI(\mathcal{A}^*) - k\epsilon \tag{2.38}$$

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.

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

Algorithm 1: Greedy/Naive Algorithm

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 δ_y and will, therefore, be less desirable. Therefore if we maintain a priority queue with the ordered values of δ_y at each step we will take the top values and update them to the current value successively. If the current value needs to be updated and the location is close to the previous optimum, it would have a small δ_y and be send back in the queue. If we meet a point which has been updated and is still a the top of the queue, it means that this point is our new optimum. This technique can be efficiently applied using **binary heaps**. This algorithm is described in the algorithm 2.

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 of using sparse kernels for the estimating the covariance matrix.

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

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;
    δ_{y^*} is updated with 2.37
    $current_y \leftarrow$ True
  end
  $\mathcal{A} \leftarrow \mathcal{A} \cup y^*$
end

Algorithm 2: Lazy Algorithm

the set $\mathcal{B}$, the set of elements $x \in \mathcal{S}$ such that $|\mathcal{K}(y,x)| > \epsilon$ we define the new set $\tilde{\mathcal{B}_y} = N(y, \epsilon)$.

This set is defined such as it contains less than d locations : $|N(y,\epsilon)| \leq d$. The truncated covariance associated with this set is named : $\tilde{K}_{y\mathcal{B}}$. Finally, we define the approximate conditional entropy $\tilde{H}_\epsilon(y|\mathcal{X}) \simeq H(y|\mathcal{X})$, computed with truncated covariance of the points of $N(y,\epsilon)$. The δ_y values are initialised by taking the difference between the true entropy and the truncated covariance, as :

$$\delta_y = \tilde{H}_\epsilon(y|\mathcal{A}) - \tilde{H}_\epsilon(y|\bar{\mathcal{A}}) \tag{2.39}$$
$$= H(y) - \tilde{H}_\epsilon(y|\mathcal{V}\backslash y) \tag{2.40}$$

Or equivalently by keeping the previous notations:

$$\delta_y = \frac{K_{yy}}{K_{yy} - \tilde{K}_{y\mathcal{V}\backslash y}\tilde{K}^{-1}_{\mathcal{V}\backslash y\mathcal{V}\backslash y}\tilde{K}_{\mathcal{V}\backslash yy}} \tag{2.41}$$

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

$$\delta_y = \tilde{H}_\epsilon(y|\mathcal{A}) - \tilde{H}_\epsilon(y|\bar{\mathcal{A}}) \tag{2.42}$$
$$= \frac{K_{yy} - \tilde{K}_{y\mathcal{A}}\tilde{K}^{-1}_{\mathcal{A}\mathcal{A}}\tilde{K}_{\mathcal{A}y}}{\tilde{K}_{yy} - \tilde{K}_{y\bar{\mathcal{A}}}\tilde{K}^{-1}_{\bar{\mathcal{A}}\bar{\mathcal{A}}}\tilde{K}_{\bar{\mathcal{A}}y}} \tag{2.43}$$

Furthermore, we also decide to update only the points that a not very correlated with the current optimal (previously placed sensor). This is justified by the fact that correlated points share a lot a mutual information and that the next best point is

Data: Covariance matrix $K_{\mathcal{VV}}$, k, $\mathcal{V}$, ϵ
Result: Sensor Selection $\mathcal{A}$
initialisation;
$\mathcal{A} = \emptyset$
foreach $y \in \mathcal{S}$ **do** $\delta_y \leftarrow 2.41$;
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** $\delta_y \leftarrow 2.43$;
  end
end

Algorithm 3: Local Kernel Algorithm

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.

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

2.5 Variational Data Assimilation

Explain Data Assimilation

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 validation 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 A.

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	

Table 3.1: Library Informations

Make sure that the packages are all listed

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).

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 discreet set points, each having a 3D coordinate and organised as an unstructured mesh. At the low center of the space the points density is very high compared to the rest of the space. Furthermore, the simulation is realised at time resolution of $\Delta t = 0.5s$ for 988 time steps or samples.

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 the figures 3.1 and 3.2

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

We can also visualise the quantity of zero elements present in the data *in function of the time*. We count every point of data that has a value over $\tau = 10^{-12}$. This gives us the plot of figure 3.3. 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 3.1: Tracer Field : Horizontal cut at $z = 1m$ and $t = 50$

Figure 3.2: Tracer Field : Horizontal cut at $z = 1m$ and $t = 988$

3.3 Preselection of the Data

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

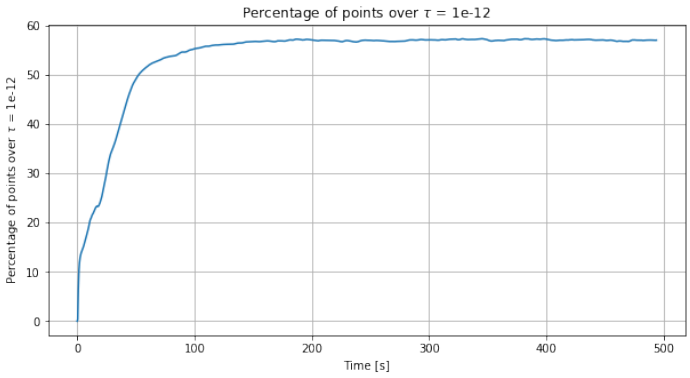

Figure 3.3: Percentage of significant points in function of the time for $\tau = 10^{-12}$

3.3.1 Selection of a working subset of the data

The first approach that we have in order to reduce the number of the 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 **??**). The resulting data shows that most of the space is unaffected by this pollutant and so we 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 $\{\mathcal{D}_k\}_{k=0}^{R}$. 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 τ. We keep then every subset $\mathcal{D}_i$ that respects the condition : $C(\mathcal{D}_i) > \tau$. This condition but guarantees that the points kept in the new working subset S have a sufficient importance in the physical world. 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 example of this algorithm is now explained. For a number of bins of 25 in each dimension and 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 3.4

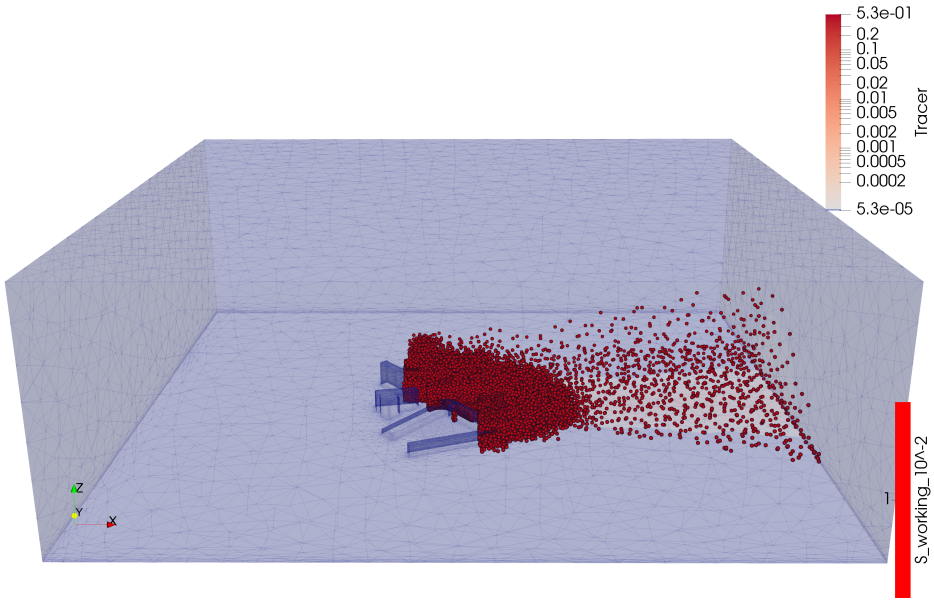

Figure 3.4: New working subset S for $\tau = 10^{-2}$

3.3.2 Selection of a subset at human level

In order 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 needs to be reached from the ground and the building top and sides for their initial placement and maintenance.

We define for each building $i \in \{1, \dots, B\}$ 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 3.5.

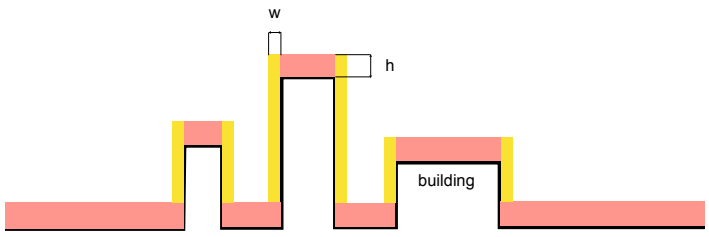

Figure 3.5: Chart Presenting the Building Profile in Human Level Selection

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 3.6) by taking the empirically the coordinates of the buildings in the small

LSBU dataset, considering a XY projection of the 3D space.

Figure 3.6: Empirical Building Shapes

Once those coordinates acquired, I used the **shapely** library (noa, 2019) in order to define the polygones associated to the buildings. In order to define the height H_i of each building I had to define a set of points overhanging it and find the minimal altitude of this point. To define the set of points I took the inner part of the surface of the building (cut by 1m) in order to avoid any edge point. This can be seen on the figure 3.7). 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

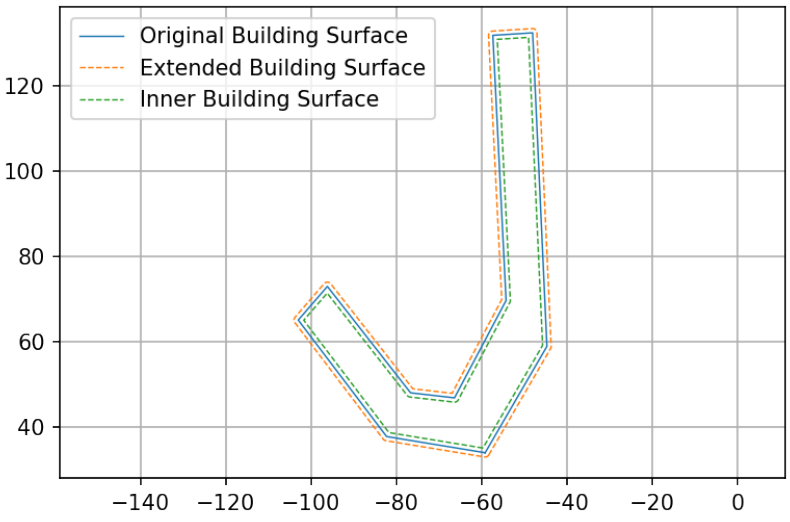

Figure 3.7: Extended an Inner building surface

Once the roof level defined we come back to the original shape of the building and enlarge it of w such as showed in figure 3.7. We then select every point of the main dataset that have XY coordinates in this extended rooftop and choose only the ones which have Z bellow the threshold $H_i + h$. This constitutes our human level data selection. An illustration can be found on figure 3.8.

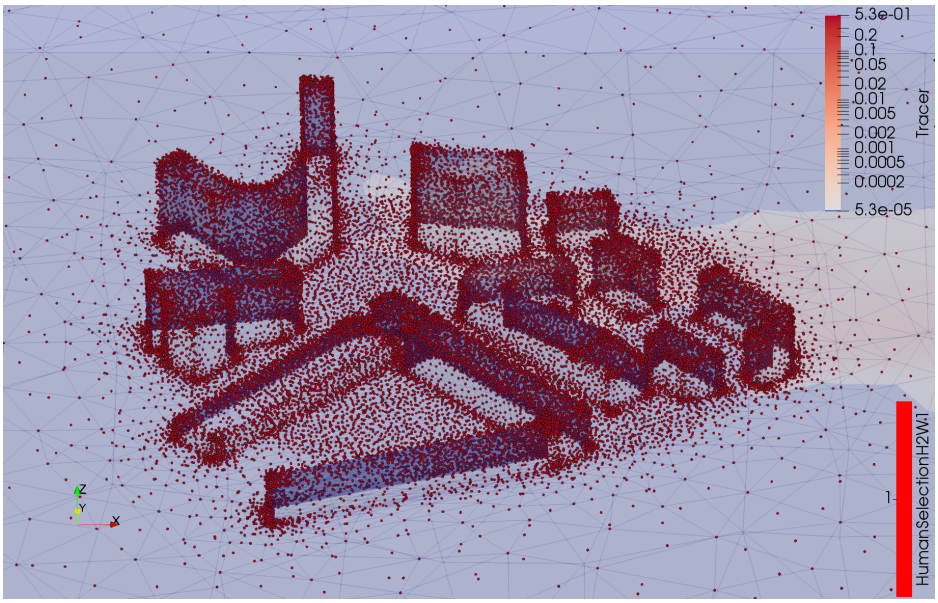

Figure 3.8: Human Level Selection

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 are able to 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 the figure 3.9 we see an illustration of the selected dataset that will be used in the rest of the project.

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. Any parameters used are explained in the chapter 4.

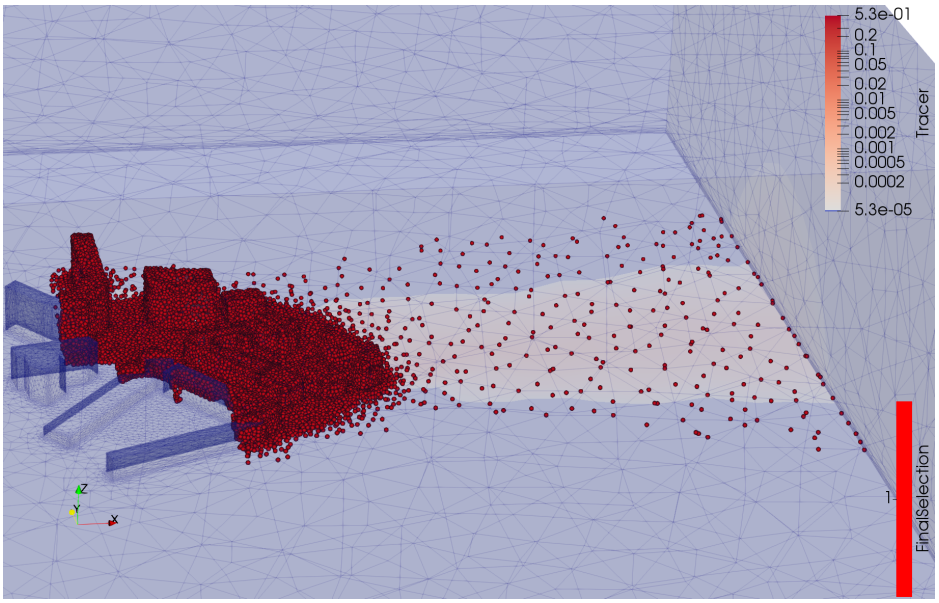

Figure 3.9: Combined Selection : Working Subset and Human Level Selection

3.5 Implementation of Gaussian Processes

GPs are in fact the main concept used in this project. We will cover here how they were implemented and what king of approximation is was made to make them more scalable.

Important to explain how the gaussian processes were implemented. Challenge of the inversion. Classical approach and TSVD approach and show simple computation times and error results.

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 parralel processing that exploits the multiple cpus of the computer in order to speed up linear algebra computations, 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 to approximate 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.

We recall, the centred data $\boldsymbol{X}$ and the sample covariance matrix $\hat{\boldsymbol{S}}$. 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 τ by keeping only τ singular values. We have the singular 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 :

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

With this approximation the inversion has a complexity reduced to $\mathcal{O}(\tau^3)$, which is a great improvement.

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.

Results add discussion on the cost of TSDV refer to the email

3.5.3 Other Approaches

A lot of literature was found 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*), I discovered that they could not be applied to our optimisation problem. Sparse GPs are in fact optimisation problems that allow to select 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 of the

loop and we 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 any sophisitcated 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 $\boldsymbol{K}_{\bar{\mathcal{A}}\bar{\mathcal{A}}}$ that is fundamentally different between each step.

Because of this, if we want to use a method that allows to remove 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.

check the last paragraph and maybe move it

3.6 Implementation of the Optimisation

Explain every algorithm implemented, their difference. This is important for the result section where sample results for those algorithm will be shown later on

3.6.1

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 :

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

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 and a lazy version of the algorithm is available at algorithm 2.

3.6.2 Comparison tools : Distance Between Sets

In order to be able to compare the results of optimisations, we define a metric that can be used to see how close 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 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 points and take the $l2$ distance between the averages ($\boldsymbol{x}(\cdot)$ being the coordinate of 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 points is actually 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 points 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 Implementation of VarDA

Write this in the end only if the validation works well

Explain why only on full set and not on comparison set

Chapter 4

Results and Analysis

After implementing the main analysis and optimisation parts we have run several experiments to find the most suitable points. For that we have explored different approaches in the estimation of 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 validated those results with 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 approaches in order to find a good covariance and we are presenting them in this section.

For this section, we present results obtained by taking the pre-selected dataset of $23'643$ 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 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't be directly used for our optimisation.

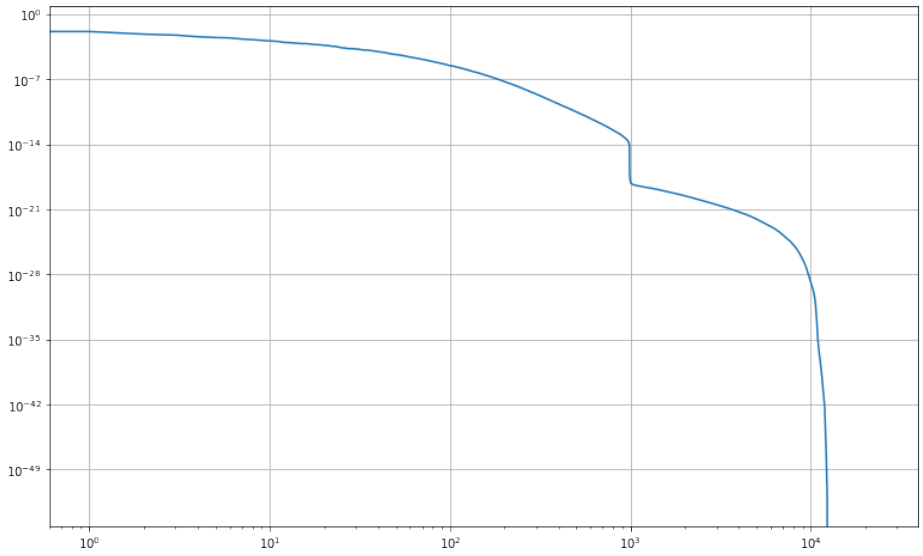

Figure 4.1: Empirical Covariance : Eigenvalues

4.1.2 Shrinkage Covariance

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

	SignDet	LogDet	λ_1	λ_p	ρ
Empirical	$(-)$	$-1'399'936.28$	0.0219	$-6.8929 \cdot 10^{-18}$	-
Shrinkage $\rho = 0.1$	$(+)$	$-351'807.64$	0.0219	$3.3603 \cdot 10^{-7}$	0.1
Shrinkage $\rho = 0.5$	$(+)$	$-314'035.58$	0.0219	$1.6801 \cdot 10^{-6}$	0.5
Ledoit-Wolf	$(+)$	$-406'320.19$	0.0219	$3.2867 \cdot 10^{-8}$	0.0097811
OAS	$(+)$	$-408'903.33$	0.0219	$2.9435 \cdot 10^{-8}$	0.0087595

Table 4.1: Shrinkage Comparison

Comment and Compare shrinkage versions

4.1.3 Arbitrary Covariance Function

Maybe not

4.2 Comparison Optimisation Algorithms

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

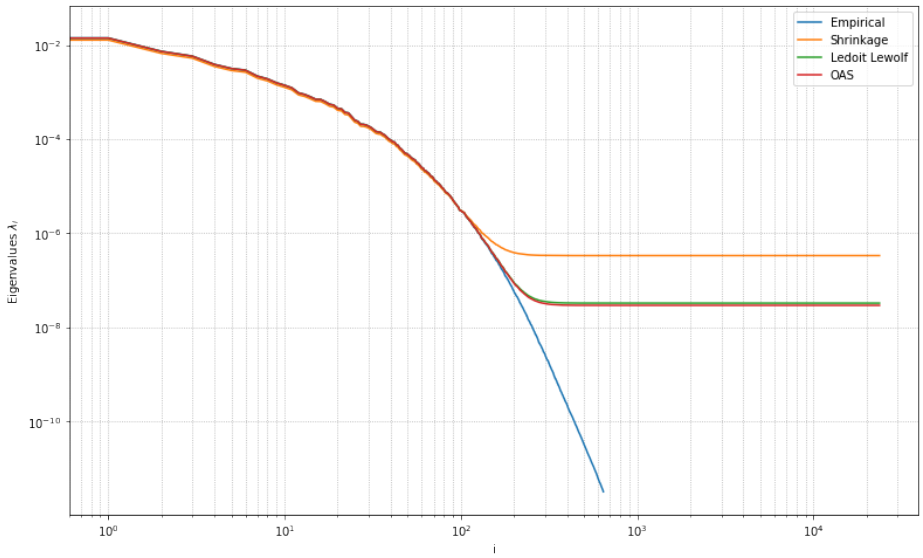

Figure 4.2: Eigenvalues Covariance Comparison

4.2.1 Conditions of the Experiment

We have defined a sphere of radius 25m, 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 XX

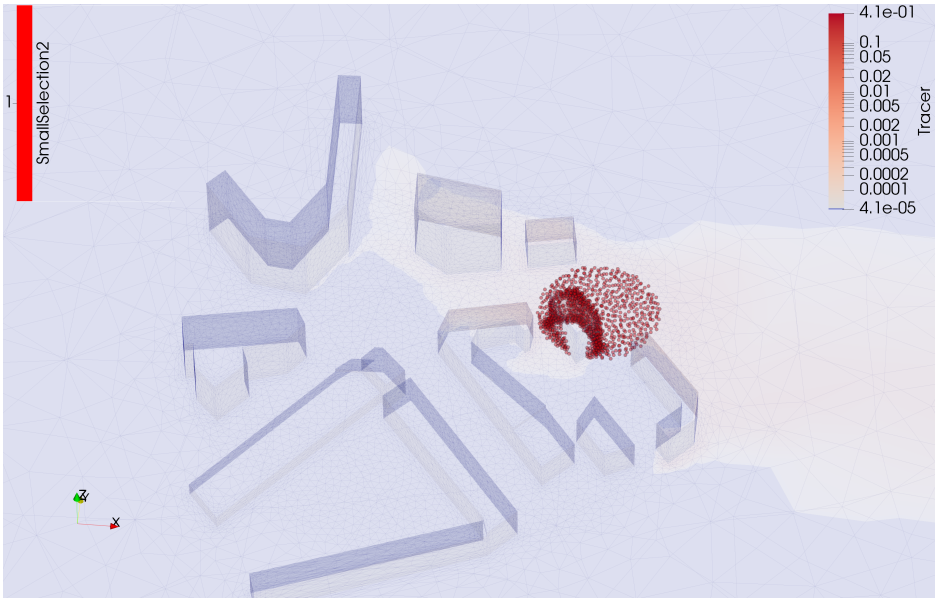

Figure 4.3: Small Subset Candidates

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 covari-

ance and the OAS estimate in figure 4.4.

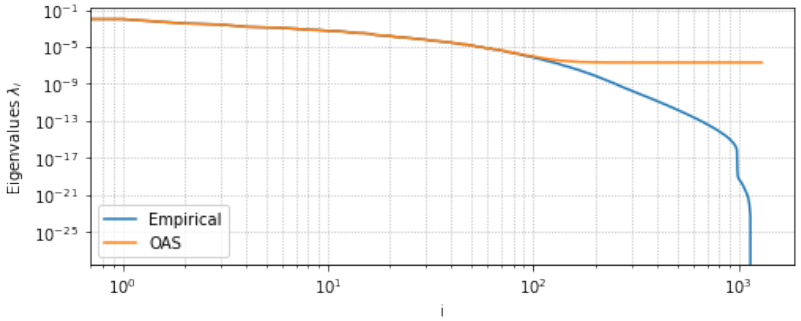

Figure 4.4: Spectrum of Empirical and OAS covariances for small dataset

Next, we are going to optimise in this space $k = 10$ points with the different algorithms and compare them based on their computation time and their 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 1043.27s and the *lazy* Optimisation gives results after 153.51s. 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.

$\mathcal{A}^*$	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 : Data

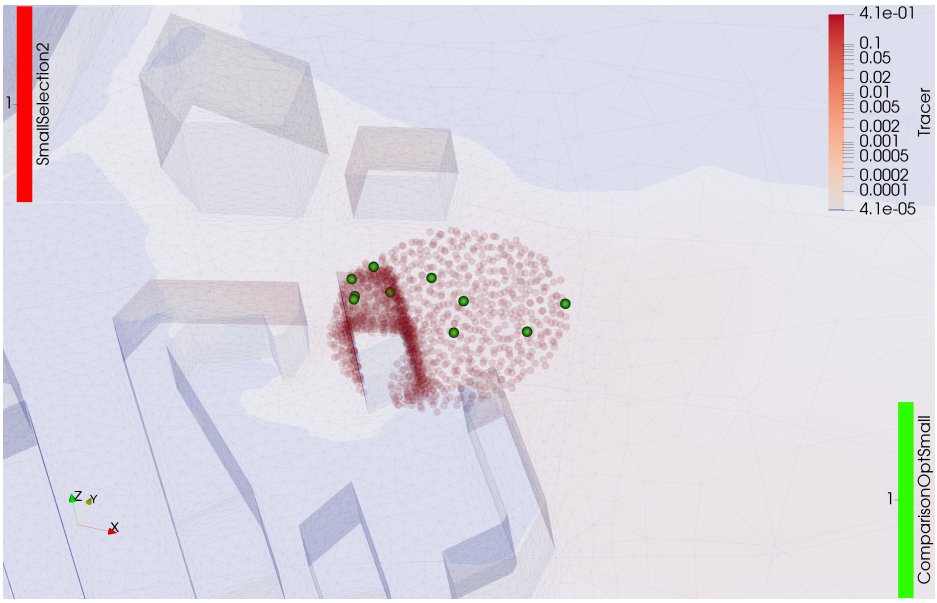

Figure 4.5: Optimal Points Location : Illustation

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 ϵ, 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 ϵ has to be chosen carefully. We are going to show that the speed, the NN distance and the parameter d varies according to ϵ, before choosing a value that will be used for the full-scale optimisation. We compute of a range of values for ϵ all those results and show them in table 4.3 and in figure 4.6.

Threshold ϵ	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

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

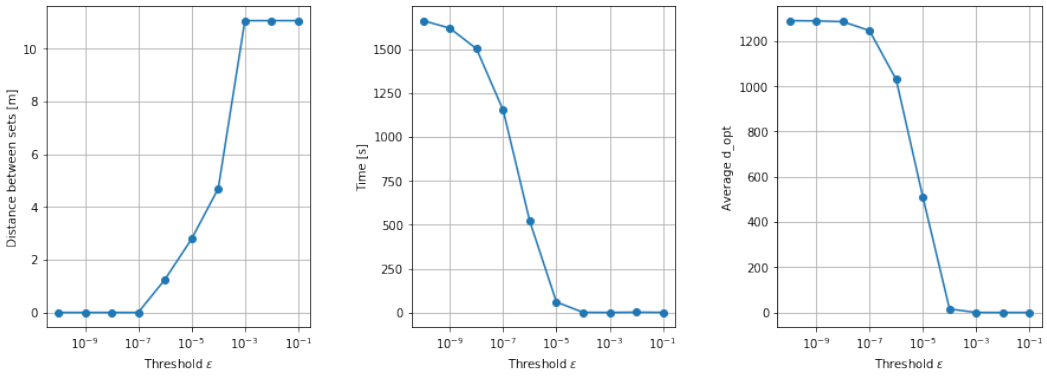

Figure 4.6: Local Kernels : Distance, Time and d, in function of ϵ

$\epsilon > 10^{-3}$, 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.

By moving 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 results. This is why the value we have chosen was optimised for an area where there is some 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 τ. 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 4.7.
As we can see the computation time increases exponentially with τ and it is then reduced when τ gets larger than $n = 988$. For the distance between the sets and the optimal results of algorithm 1, we see that it is quite low at the beginning, is reaches a minimum at $\tau_{opt} = 25$ and it then increases this the truncation parameter. We will keep this method for trying it on the full dataset in the next section.

4.2.5 Arbitrary Covariance

Alternatively, we define the covariance using an *isotropic* and *stationary* kernel function with arbitrary parameters. We chose for this experiment the 5/2 Matérn kernel. Unrelated to the data !!

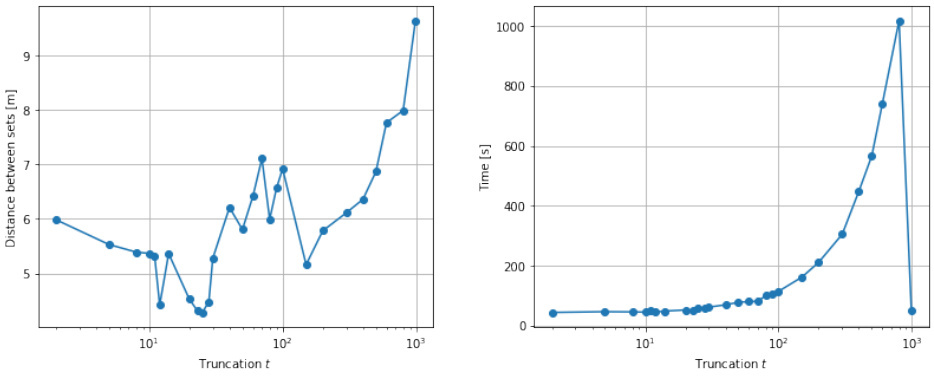

Figure 4.7: TSVD Approximation : Distance and Time in function of τ

Add in literature review // Implementation the Kernels definitions

4.3 Optimisation Results

After the analysis of different algorithm results on a small sample 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.

We visualise the position of the optimal set of sensor points $\mathcal{A}^*$ at different scales

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

in the figures 4.8, 4.9 and 4.10. 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 : Data

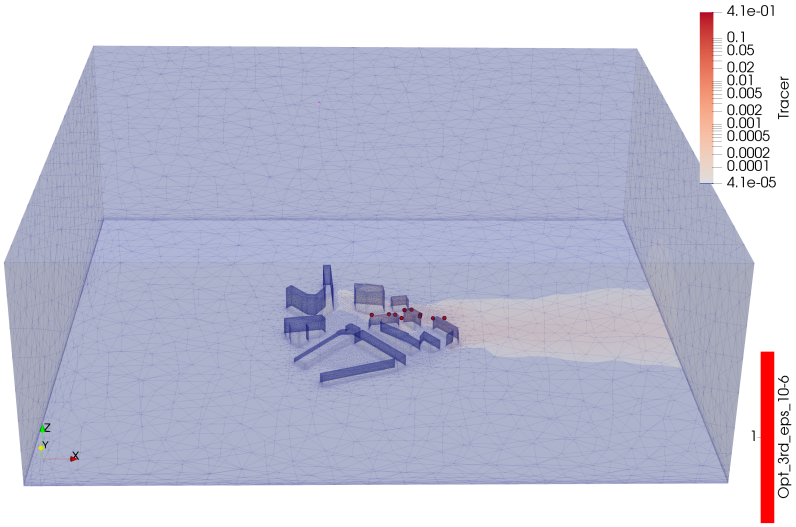

Figure 4.8: Position of the Optimal Set : Main View

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 seems to be a good thing because we have a propagation originating at the centre and with wind in the east direction.

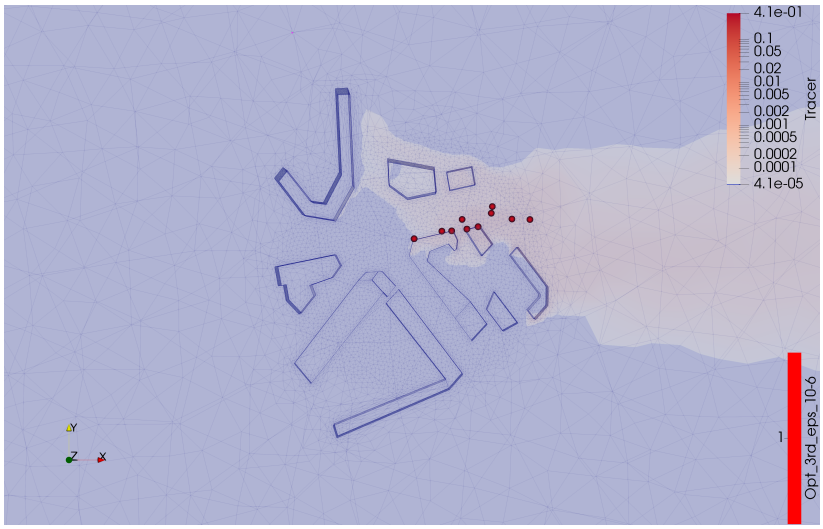

Figure 4.9: Position of the Optimal Set : Top View

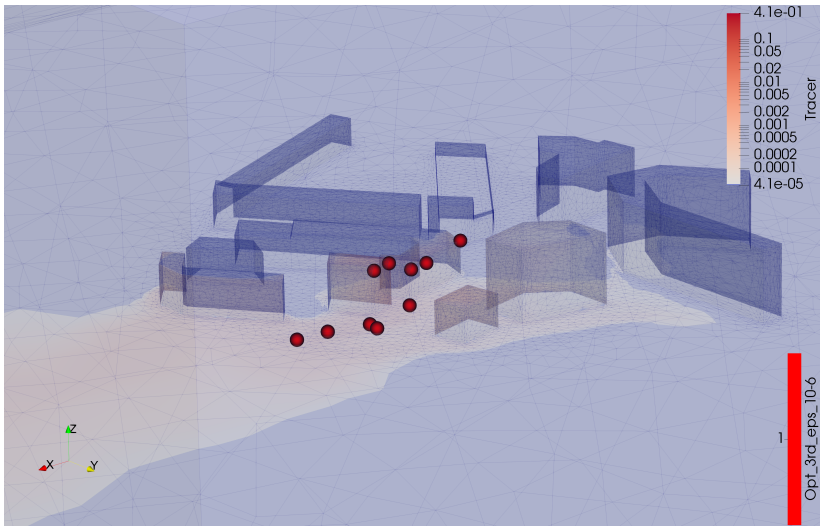

Figure 4.10: Position of the Optimal Set : Close View

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 figures 4.11 and 4.12 along with the previous optimal results.

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 at the border the the pre-selected dataset. The distance between this set and the previous one is of : 13.84m which is quite important.

$\mathcal{A}^*$	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 : Data

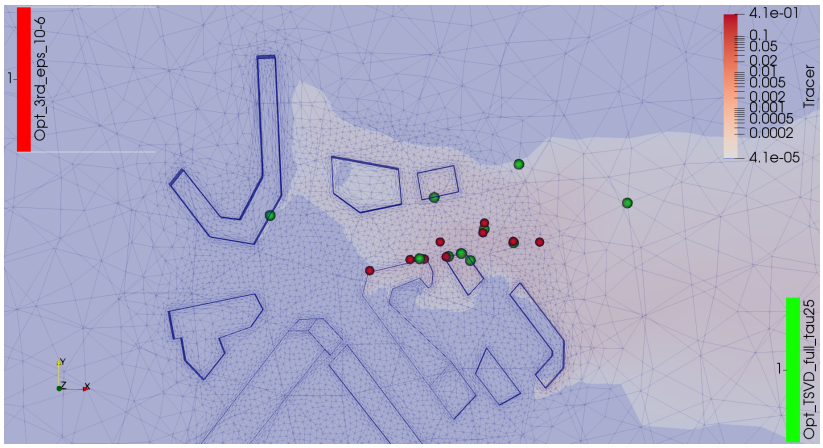

Figure 4.11: Position of the Optimal Set : Main View - $\tau = 25$

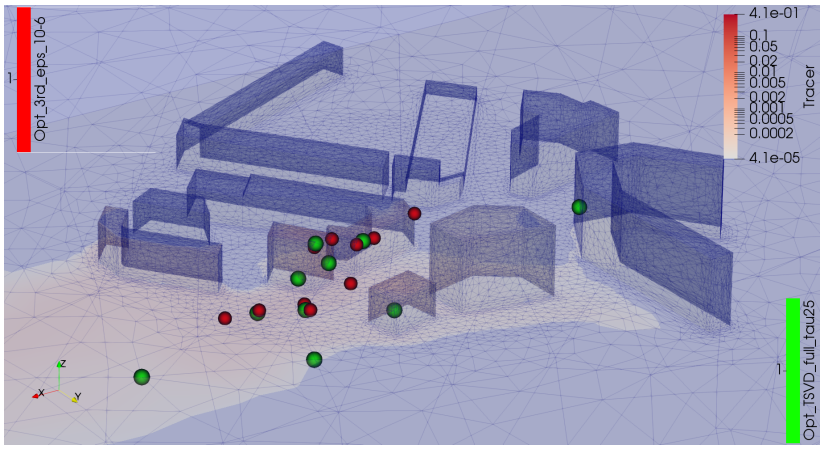

Figure 4.12: Position of the Optimal Set : Main View $\tau = 25$

4.4 Validation with VarDA

Add Validation Results

Chapter 5

Conclusion

Change Conclusion of the Project

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

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

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

Appendix A

Description of the Code

Bibliography

(2019). Shapely : Manipulation and analysis of geometric objects. Contribute to Toblerity/Shapely development by creating an account on GitHub. original-date: 2011-12-31T19:43:11Z. pages 22

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 18

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 12

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 11, 12

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 25

Krause, A., Singh, A., and Guestrin, C. (2008). Near-Optimal Sensor Placements in Gaussian Processes: Theory, Efficient Algorithms and Empirical Studies. page 50. pages 7, 8, 11, 12, 13, 14, 16, 26

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 9,

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 13

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

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

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