Master's Programme in Mathematics and Statistics

Power plant carbon dioxide emission quantification with plume modeling

Janne Nurmela

May 5, 2022

Supervisor(s): Professor Matti Lassas
Dr. Janne Hakkarainen

Examiner(s): Professor Matti Lassas

University of Helsinki
Faculty of Science

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Degree programme
Faculty of Science	Master's Programme in Mathematics and Statistics

Tekijä — Författare — Author

Janne Nurmela

Työn nimi — Arbetets titel — Title

Power plant carbon dioxide emission quantification with plume modeling

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidantal — Number of pages
Master's thesis	May 5, 2022	76

Tiivistelmä — Referat — Abstract

The quantification of carbon dioxide emissions pose a significant and multi-faceted problem for the atmospheric sciences as a part of the research regarding global warming and greenhouse gases. Emissions originating from point sources, referred to as plumes, can be simulated using mathematical and physical models, such as a convection-diffusion plume model and a Gaussian plume model.

The convection-diffusion model is based on the convection-diffusion partial differential equation describing mass transfer in diffusion and convection fields. The Gaussian model is a special case or a solution for the general convection-diffusion equation when assumptions of homogeneous wind field, relatively small diffusion and time independence are made. Both of these models are used for simulating the plumes in order to find out the emission rate for the plume source.

An equation for solving the emission rate can be formulated as an inverse problem written as $\boldsymbol{y} = F(\boldsymbol{x}) + \boldsymbol{\epsilon}$ where $\boldsymbol{y}$ is the observed data, F is the plume model, $\boldsymbol{\epsilon}$ is the noise term and $\boldsymbol{x}$ is an unknown vector of parameters, including the emission rate, which needs to be solved. For an ill-posed inverse problem, where F is not well behaved, the solution does not exist, but a minimum norm solution can be found. That is, the solution is a vector $\boldsymbol{x}$ which minimizes a chosen norm function, referred to as a loss function.

This thesis focuses on the convection-diffusion and Gaussian plume models, and studies both the difference and the sensibility of these models. Additionally, this thesis investigates three different approaches for optimizing loss functions: the optimal estimation for linear model, Levenberg–Marquardt algorithm for non-linear model and adaptive Metropolis algorithm. A goodness of different fits can be quantified by comparing values of the root mean square errors; the better fit the smaller value the root mean square error has.

A plume inversion program has been implemented in Python programming language using the version 3.9.11 to test the implemented models and different algorithms. Assessing the parameters' effect on the estimated emission rate is done by performing sensitivity tests for simulated data. The plume inversion program is also applied for the satellite data and the validity of the results is considered. Finally, other more advanced plume models and improvements for the implementation will be discussed.

ACM Computing Classification System (CCS):
Applied computing → Physical sciences and engineering → Earth and atmospheric sciences → Environmental sciences

Avainsanat — Nyckelord — Keywords

plume simulation, plume inversion, inverse problems, gaussian plume, convection-diffusion

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

This thesis was written in collaboration with the Finnish Meteorological Institute.

Contents

List of Symbols

$\boldsymbol{I}$	Unit matrix of arbitrary dimensions
$\boldsymbol{O}$	Zero matrix of arbitrary dimensions
$\boldsymbol{r}$	Residual vector between the model and the data
$\boldsymbol{v}$	Vector or vector-valued function describing wind
$\boldsymbol{\Sigma}_{\text{obs}}$	Observation covariance matrix
$\boldsymbol{\Sigma}_{\text{pos}}$	Posterior covariance matrix
$\boldsymbol{\Sigma}_{\text{pri}}$	Prior covariance matrix
$\boldsymbol{\Sigma}_{\boldsymbol{x}}$	Symmetric positive-definite covariance matrix of vector $\boldsymbol{x}$
$\delta(x)$	Delta distribution centered at x
$\mathcal{L}$	Real-valued loss function to be minimized during optimization
$\mathcal{N}$	Normal distribution
∇	Nabla operator in Cartesian coordinate system
∇^2	Laplace operator in Cartesian coordinate system
$\|\|\boldsymbol{x}\|\|_p$	l^p-norm of a vector $\boldsymbol{x}$
π_{pos}	Posterior distribution
π_{pri}	Prior distribution
τ	Temporal discretization
$\text{corr}_{\boldsymbol{x}}$	Correlation matrix of vector $\boldsymbol{x}$
a	Atmospheric stability parameter
b	Exponent of the standard deviation term in the Gaussian plume model
c	Real-valued function describing concentration
$C^k(\Omega)$	Space of real valued k-times continuously differentiable functions in $\Omega \subset \mathbb{R}^n$
C_D	Diffusion scale defining a relation to the wind speed
D	Real number or a real-valued function describing diffusion

H	Hessian matrix
h	Spatial discretization
J	Jacobian matrix
L	Likelihood function of data given model parameters
Q	Real number describing the source rate
R	Radius of earth, approximately 6371 km
S	Real-valued function describing sources
u	Average wind speed in plume region
VC_{gas}	Total vertical column of gas
X_{gas}	Column average of gas

1. Introduction

Global warming has become one of the greatest threats for modern societies and will have a great impact to the lives of future generations. The main cause for this global climate change are greenhouse gases emitted into the Earth's atmosphere by human activity. Traffic and industrial activities such as power plants keep increasing the overall amount of greenhouse gases in the atmosphere. Due to relevance of the global warming, monitoring greenhouse gas emissions has become a noteworthy research topic among atmospheric sciences. This research includes modeling how greenhouse gas emissions behave after ejected into the atmosphere.

In the United Nations Climate Change Conference held on 12.12.2015 in Paris — shortened as the COP 21 meeting — an agreement on limiting the global warming below 2 degrees Celsius was adopted. This agreement is usually referred to as "The Paris Agreement on Climate Change" [23]. As a consequence, many countries are required to reduce their greenhouse gas emissions as soon as possible. These reduction measures include monitoring and lowering emissions produced by power plants as they are a major greenhouse gas emission source, alongside traffic of cars and ships. Emissions reported by the power plants may not always be truthful due to financial and economical gains which encourages scientists to develop methods to quantify factory emissions using independent data.

This thesis focuses on quantifying CO_2 emissions originated from chimneys. An emission originated from a point source is called a plume and it can be observed from space using satellites, e.g. using Orbiting Carbon Observatory-2 (OCO-2) [3]. An estimation for the magnitude of the plume source is done via a mathematical inversion: the plume is simulated using a plume model with emission rate as an input parameter and then compared to the observed satellite data. An approximation for the emission rate is the emission rate which results in the best representation of the observed data after the plume model is applied.

In this thesis two different plume models, a Gaussian plume model and a convection-diffusion plume model, are used for inversion to find out a magnitude of the emission source. Tests for the sensitivity to the input parameters, pixel size, performance and other aspects regarding these models are also be discussed. The models and their numerical

implementations are introduced in Chapter 2. Chapter 3 introduces basic knowledge and theory for inverse problems and Chapter 4 presents briefly the external data products used for simulating the plumes, as well as a method for interpolating data. Chapter 5 shows and discuss the results when the plume inversion program is applied on different datasets. The used data includes both synthetic and real world observations collected by the OCO-2 satellite. Finally, Chapter 6 discusses the used methods, current issues and further improvements to the plume inversion program.

2. Plume models

This chapter focuses on covering plume models which are used for carbon dioxide plume inversion. These models are essentially mass transport models which describe how particles behave when affected by convection and diffusion. The first model is the convection-diffusion model described in Section 2.1 which is a general partial differential equation describing particle movement. The second model is called Gaussian plume model given in Section 2.2 which is a special case of the general convection-diffusion model with certain conditions assumed.

2.1 Convection-diffusion model

The convection-diffusion plume model is based on the convection-diffusion equation which is a general equation describing mass or heat transfer in given convection and diffusion fields. The convection-diffusion equation is a partial differential equation which is a combination of diffusion equation and linear transport equation [9].

2.1.1 Formulation

Define a non-negative function $c\colon \mathbb{R}^n \times \mathbb{R}_+ \to [0, \infty)$, where $\mathbb{R}_+ = (0, \infty)$, which tells the mass concentration at given location $\boldsymbol{x} \in \mathbb{R}^n$ and at time $t \in \mathbb{R}_+$. The general convection-diffusion equation can be formulated as PDE (partial differential equation) by the formula

$$\partial_t c(\boldsymbol{x}, t) = \nabla \cdot (D(\boldsymbol{x}, t)\nabla c(\boldsymbol{x}, t)) - \nabla \cdot (\boldsymbol{v}(\boldsymbol{x}, t)c(\boldsymbol{x}, t)) + S(\boldsymbol{x}, t), \tag{2.1}$$

where D is a diffusivity describing diffusion rate at each location, $\boldsymbol{v}$ is a velocity field describing convection at each location, ∇ is the nabla operator operating with respect to spatial variable $\boldsymbol{x}$ and S describes the sources and/or sinks. The function c is assumed to be $C^2(\mathbb{R}^n)$ with respect to the spatial variable and $C^1(\mathbb{R}_+)$ with respect to the temporal variable. The diffusion is described by a scalar field and the convection described by a vector field. The diffusion in this thesis is not a natural diffusion of a gas but rather a turbulent diffusion caused by the wind field.

Equation (2.1) can be written in a different form by using the product rule for divergence as

$$\nabla \cdot (D(\boldsymbol{x},t)\nabla c(\boldsymbol{x},t)) = \nabla D(\boldsymbol{x},t) \cdot \nabla c(\boldsymbol{x},t) + D(\boldsymbol{x},t)\nabla^2 c(\boldsymbol{x},t),$$
$$\nabla \cdot (\boldsymbol{v}(\boldsymbol{x},t)c(\boldsymbol{x},t)) = (\nabla \cdot \boldsymbol{v}(\boldsymbol{x},t))c(\boldsymbol{x},t) + \boldsymbol{v}(\boldsymbol{x},t) \cdot \nabla c(\boldsymbol{x},t).$$

An usual assumption is that the vector field $\boldsymbol{v}$ has a zero-divergence or $\nabla \cdot \boldsymbol{v}(\boldsymbol{x},t) = 0$. Although numerically the divergence of the velocity may not be exactly zero, the effect of the divergence is insignificant compared to the convection due to wind. Similarly, the inner product between the gradient of the diffusion field and the gradient of the concentration is assumed to relatively small effect so $\nabla D(\boldsymbol{x},t) \cdot \nabla c(\boldsymbol{x},t) \approx 0$. The source term S describes point sources and in this thesis each point source is assumed to be constant in time. With these assumptions, equation (2.1) becomes

$$\partial_t c(\boldsymbol{x},t) = D(\boldsymbol{x},t)\nabla^2 c(\boldsymbol{x},t) - \boldsymbol{v}(\boldsymbol{x},t) \cdot \nabla c(\boldsymbol{x},t) + S(\boldsymbol{x}). \tag{2.2}$$

Usually the goal is to find a steady state solution which describes a state where the time dependency vanishes. If the diffusion and wind fields are smooth and regular enough, then the solution for (2.2) will reach the steady state as $t \to \infty$. In practice, this means that the diffusion, the wind and the source term balance each other out after enough time has passed. The steady state solution satisfies the equation

$$0 = D(\boldsymbol{x})\nabla^2 c(\boldsymbol{x}) - \boldsymbol{v}(\boldsymbol{x}) \cdot \nabla c(\boldsymbol{x}) + S(\boldsymbol{x}), \tag{2.3}$$

which is the same equation as (2.2) but without time dependency. In reality, both the diffusion and wind fields change in time, which makes finding the steady state solution impossible. However, Equation (2.2) can be simulated numerically long enough such that it reaches near stable state. It is worth noting that even with exact forms for diffusion and wind fields, both of the partial differential equations (2.2) and (2.3) are generally impossible to solve in analytic form.

The parameters which can be fitted for the convection-diffusion equation are diffusion emission rate and diffusion scale. Diffusion scale is a scalar factor between the wind speed and the diffusion field at each location introduced later in condition (2.10) in the next section. Although it may not be obvious, the solution satisfying (2.3) is in fact linear with respect to the source rate. That is, the values of the steady state solution are directly proportional to the source rate and separate sources are independently additive to the total concentration. Linearity with respect to the source rate is assumed without further consideration.

2.1.2 Numerical implementation

The goal is to find a way to numerically simulate the equation (2.2). This means that the next state after time step τ can be computed by using the current state and a zero state zero will act as an initial state. The wind field changes in time but will not be updated every time step as it would be computationally too heavy due to updating the wind field requiring an interpolation and rotation. The diffusion will be directly depended on the wind field so also the diffusion field will be updated as the wind field updates. The data product used for the wind field data is presented in Chapter 4.

A numerical computation of the convection-diffusion equation involves derivatives which are approximated via a method called finite differences. The idea of the method is to approximate any differential operator at a point by using nearby known values. The chosen number of nearby points used for approximating the derivative is referred as a stencil. For example, three-point stencil means that nearest three points are used for the derivative approximation.

Mathematically, the derivative of a function $f\colon \mathbb{R} \to \mathbb{R}$ is defined as a limit

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

if the limit exists. For h small enough the derivative can be numerically approximated as a forward difference

$$f'_+(x) \approx \frac{f(x+h) - f(x)}{h}.$$

or as a backward difference

$$f'_-(x) \approx \frac{f(x) - f(x-h)}{h}.$$

By taking the average of the forward and backward differences, the derivative can be approximated as a central difference

$$f'(x) = \frac{f'_+(x) + f'_-(x)}{2} \approx \frac{-f(x-h) + f(x+h)}{2h}. \tag{2.4}$$

For a smooth function, the smaller step size give a better approximation. However, the parameter h can not be arbitrary small as computers have a finite precision (there exists the smallest non-zero positive number often referred to as the machine epsilon).

In this thesis, derivatives are required to be approximated using known values in a regular grid and h is the step size (discretization) used in the grid. The central difference (2.4) can be written as a difference matrix

$$\frac{1}{2h} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}.$$

which means that the gird values $f(x_{i-1}), f(x_i), f(x_{i+1})$ are required to compute an approximation for $f'(x_i)$. The central difference approximation is then

$$f'(x_i) \approx \frac{1}{2h} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} f(x_{i-1}) & f(x_i) & f(x_{i+1}) \end{bmatrix}^T = \frac{-f(x_{i-1}) + f(x_{i+1})}{2h}.$$

where h is step size between points x_i.

The approximation for the second derivative can be done in a similar fashion. The second central difference is given by

$$f''(x) \approx \frac{\frac{f(x+h) - f(x)}{h} - \frac{-f(x-h) + f(x)}{h}}{h} = \frac{f(x-h) - 2f(x) + f(x+h)}{h^2},$$

which can be written in the matrix form as

$$\frac{1}{h^2} \begin{bmatrix} 1 & -2 & 1 \end{bmatrix}.$$

Again, this corresponds the computation

$$f''(x_i) \approx \frac{1}{h^2} \begin{bmatrix} 1 & -2 & 1 \end{bmatrix} \cdot \begin{bmatrix} f(x_{i-1}) & f(x_i) & f(x_{i+1}) \end{bmatrix}^T = \frac{f(x_{i-1}) - 2f(x_i) + f(x_{i+1})}{h^2}.$$

Both of the approximations for f' and f'' are computed using three-point stencil but larger number of points could also be used.

The finite differences can be generalized to higher dimensions. In 2D, matrices corresponding differential operators are two-dimensional where rows corresponds y-axis and columns x-axis. In 2D, the partial derivative matrices are

$$\partial_x = \begin{bmatrix} 0 & 0 & 0 \\ -\frac{1}{2h_x} & 0 & \frac{1}{2h_x} \\ 0 & 0 & 0 \end{bmatrix} \text{ and } \partial_y = \begin{bmatrix} 0 & \frac{1}{2h_y} & 0 \\ 0 & 0 & 0 \\ 0 & -\frac{1}{2h_y} & 0 \end{bmatrix},$$

and the second partial derivative matrices are

$$\partial_x^2 = \begin{bmatrix} 0 & 0 & 0 \\ \frac{1}{h_x^2} & -\frac{2}{h_x^2} & \frac{1}{h_x^2} \\ 0 & 0 & 0 \end{bmatrix} \text{ and } \partial_y^2 = \begin{bmatrix} 0 & \frac{1}{h_y^2} & 0 \\ 0 & -\frac{2}{h_y^2} & 0 \\ 0 & \frac{1}{h_y^2} & 0 \end{bmatrix}.$$

Similarly in 3D, the partial derivative matrices are

$$\begin{array}{lccc} & \text{First layer} & \text{Second layer} & \text{Third layer} \\ \partial_x = & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, & \begin{bmatrix} 0 & 0 & 0 \\ -\frac{1}{h_x^2} & 0 & \frac{1}{h_x^2} \\ 0 & 0 & 0 \end{bmatrix}, & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \\ \partial_y = & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, & \begin{bmatrix} 0 & \frac{1}{h_y^2} & 0 \\ 0 & 0 & 0 \\ 0 & -\frac{1}{h_y^2} & 0 \end{bmatrix}, & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \\ \partial_z = & \begin{bmatrix} 0 & 0 & 0 \\ 0 & -\frac{1}{h_z^2} & 0 \\ 0 & 0 & 0 \end{bmatrix}, & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, & \begin{bmatrix} 0 & 0 & 0 \\ 0 & \frac{1}{h_z^2} & 0 \\ 0 & 0 & 0 \end{bmatrix}, \end{array}$$

and the second partial derivative matrices are

$$\begin{array}{lccc} & \text{First layer} & \text{Second layer} & \text{Third layer} \\ \partial_x^2 = & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, & \begin{bmatrix} 0 & 0 & 0 \\ \frac{1}{h_x^2} & -\frac{2}{h_x^2} & \frac{1}{h_x^2} \\ 0 & 0 & 0 \end{bmatrix}, & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \\ \partial_y^2 = & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, & \begin{bmatrix} 0 & \frac{1}{h_y^2} & 0 \\ 0 & -\frac{2}{h_y^2} & 0 \\ 0 & \frac{1}{h_y^2} & 0 \end{bmatrix}, & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \\ \partial_z^2 = & \begin{bmatrix} 0 & 0 & 0 \\ 0 & \frac{1}{h_z^2} & 0 \\ 0 & 0 & 0 \end{bmatrix}, & \begin{bmatrix} 0 & 0 & 0 \\ 0 & -\frac{2}{h_z^2} & 0 \\ 0 & 0 & 0 \end{bmatrix}, & \begin{bmatrix} 0 & 0 & 0 \\ 0 & \frac{1}{h_z^2} & 0 \\ 0 & 0 & 0 \end{bmatrix}. \end{array}$$

In practice, plumes evolve in three spatial dimensions, which are longitude, latitude and altitude, corresponding to x-, y- and z-directions respectively. However, as simulations in three dimensions can easily become computationally too heavy, the simulations will be done in two dimensions. The altitude (z-axis) dimension can be removed by properly averaging over different altitudes, as discussed in Appendix A. The two remaining dimensions are longitude and latitude for which a common discretization $h_x = h_y = h$ will be used. This discretization of the longitude and latitude grids can usually vary from $h = 0.001°$ to $h = 0.1°$. The used discretization and the computation time naturally correlate; larger grids imply larger matrices and heavier matrix products to compute.

By using a shared discretization h for x and y directions, the first and second partial

derivative matrices are

$$\partial_x = \frac{1}{2h}\begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \text{ and } \partial_y = \frac{1}{2h}\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}. \tag{2.5}$$

$$\partial_x^2 = \frac{1}{h^2}\begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix} \text{ and } \partial_y^2 = \frac{1}{h^2}\begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix}. \tag{2.6}$$

By summing the second partial derivative matrices (2.6) together gives a standard five-point stencil approximation in 2D for the Laplacian as

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} = \frac{1}{h^2}\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

That is, in order to compute a numerical Laplacian for a function $f\colon \mathbb{R}^2 \to \mathbb{R}$ at (x_i, y_j), the function evaluations at grid points

$$(x_i, y_{j-1}), (x_{i-1}, y_j), (x_i, y_j), (x_{i+1}, y_j), (x_i, y_{j+1})$$

are required. An approximation for the Laplacian can be written as a matrix product by reshaping matrices into 1D vectors. By defining a (column) vector of all nearby values of f as

$$\boldsymbol{f} = \begin{bmatrix} f(x_i, y_{j-1}) & f(x_{i-1}, y_j) & f(x_i, y_j) & f(x_{i+1}, y_j) & f(x_i, y_{j+1}) \end{bmatrix}^{\mathrm{T}}$$

and defining the Laplacian reduced to 1D as a (row) vector

$$\nabla^2 = \frac{1}{h^2}\begin{bmatrix} 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \end{bmatrix}$$

enables the approximation of the Laplacian at (x_i, y_j) to be computed as a matrix product

$$\nabla^2 f(x_i, y_j) \approx \nabla^2 \cdot \boldsymbol{f} = \frac{f(x_i, y_{j-1}) + f(x_{i-1}, y_j) - 4f(x_i, y_j) + f(x_{i+1}, y_j) + f(x_i, y_{j+1})}{h^2}.$$

To get more accurate Laplacian, a nine-point stencil can be used for the Laplacian which also inludes the corners. The weighting factors for the corners are computed by weighting the corners with their corresponding Euclidean distance to the center point. In the paper by O'Reilly and Beck [17], weights for a 27-point discrete Laplacian in 3D are computed in detail. That is, a Laplacian matrix includes every point inside 3×3 cube. By following the same idea for 2D case, a nine-point Laplacian could be derived. However, in 2D case a nine-point Laplacian can be derived with a simpler computation based on a rotation of the standard stencil as shown in the proof of Theorem 2.1.

Theorem 2.1. *A nine-point Laplacian with grid dicretization h in x and y directions is given by*

$$\nabla^2 = \frac{1}{4h^2}\begin{bmatrix} 1 & 2 & 1 \\ 2 & -12 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \frac{1}{h^2}\begin{bmatrix} 1/4 & 1/2 & 1/4 \\ 1/2 & -3 & 1/2 \\ 1/4 & 1/2 & 1/4 \end{bmatrix}.$$

The nine-point Laplacian can be though as being the average between the standard five-point stencil and a rotated five-point stencil. The values in both stencils are weighted by the square of the Euclidean distance to the center.

Proof. The standard stencil, in which points other than the center have a square distance h^2 to the center, is

$$\frac{1}{h^2}\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

as presented before. The rotated stencil, in which points other than the center have a square distance $2h^2$ to the center, is

$$\frac{1}{2h^2}\begin{bmatrix} 1 & 0 & 1 \\ 0 & -4 & 0 \\ 1 & 0 & 1 \end{bmatrix} = \frac{1}{h^2}\begin{bmatrix} 1/2 & 0 & 1/2 \\ 0 & -2 & 0 \\ 1/2 & 0 & 1/2 \end{bmatrix}.$$

Taking the average of these stencils yields to the nine-point stencil as

$$\nabla^2 = \frac{1}{2}\cdot\frac{1}{h^2}\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} + \frac{1}{2}\cdot\frac{1}{h^2}\begin{bmatrix} 1/2 & 0 & 1/2 \\ 0 & -2 & 0 \\ 1/2 & 0 & 1/2 \end{bmatrix} = \frac{1}{h^2}\begin{bmatrix} 1/4 & 1/2 & 1/4 \\ 1/2 & -3 & 1/2 \\ 1/4 & 1/2 & 1/4 \end{bmatrix}.$$

□

A two-dimensional grid representation with shared discretization h for both dimensions is illustrated in Figure 2.1. Points typically used for approximating differential operators are colored. Arbitrary number of surrounding grid points could be used for approximating the derivative but in this thesis, only nearby pixels are used.

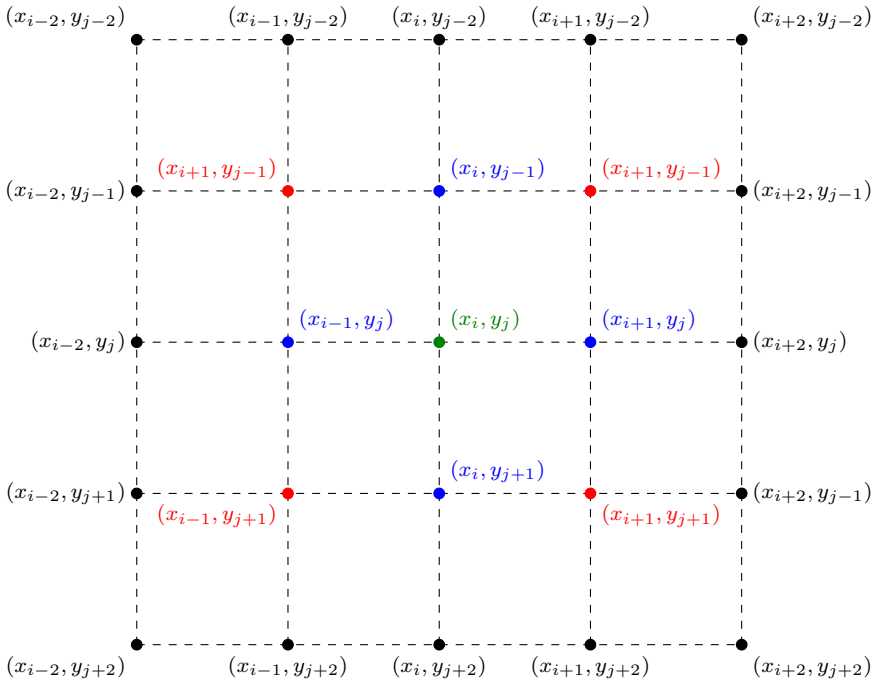


Figure 2.1: Typical points used in a small stencil on a two-dimensional grid with a shared grid discretization. The center point is green, face points are blue and corner points are red. For small stencil approximations, black points are not used.

Computing a differential operator approximation simultaneously at every point for a function in a grid requires a typically large difference matrix. If a function $f\colon \mathbb{R}^2 \to \mathbb{R}$ is defined in three-dimensional grid of size $N_y \times N_x$, then the values are reduced to a 1D vector of length $N_y N_x$. Consequently, a difference matrix for computing difference operator approximation at each grid point has then shape $N_y N_x \times N_y N_x$ which can easily become too large if the used discretization is too dense.

Each row of the difference matrix has non-zero elements equal to the stencil size used for approximation. The first row of the difference matrix multiplied with value vector $\boldsymbol{f}$ gives the difference operator approximation at the first grid point, the second row multiplied with $\boldsymbol{f}$ gives an approximation of the difference operator at the second grid point and so on.

As an example, consider a partial operator ∂_x for a function $f\colon \mathbb{R}^2 \to \mathbb{R}$ defined in $N_y \times N_x$ regular grid with discretization h. The difference matrix has shape (dimension) $N_y N_x \times N_y N_x$ and has values $-1/(2h)$ below diagonal and values $1/(2h)$ above diagonal, representing central differences. As the left boundary has no value to the left and the right boundary has no value to the right, boundary values are kept as zero. That is, every time the point for which the central difference would be computed is on the boundary, the corresponding row in the matrix consists of zeros. This can be characterized by a

$N_x \times N_y N_x$ zero matrix

$$\boldsymbol{O} = \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix},$$

which ensures that every value in the first and last row of the $N_y \times N_x$ grid remains zero. The actual (central) difference matrix is a $N_x(N_y - 2) \times N_y N_x$ matrix given by

$$\boldsymbol{D} = \frac{1}{2h} \begin{bmatrix} 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & -1 & 0 & 1 & 0 \\ 0 & \dots & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & \dots & 0 & 0 & 0 \end{bmatrix},$$

where again the row consists of zeros if the point is in the first or last column of the $N_y \times N_x$ grid. The matrix $\boldsymbol{D}$ computes the partial derivative using central difference for each non-boundary point by using values to the left and right. The final difference matrix for operator ∂_x in regular $N_y \times N_x$ grid is $N_y N_x \times N_y N_x$ matrix given by a stacked $N_y N_x \times N_y N_x$ matrix

$$\partial_x = \frac{1}{2h} \begin{bmatrix} \boldsymbol{O} \\ \boldsymbol{D} \\ \boldsymbol{O} \end{bmatrix},$$

The next goal is to find a similar matrix representations for operators in Equation (2.2) which in 2D has a form

$$\partial_t c(x, y, t) = [D(x, y)\nabla^2 - \boldsymbol{v}(x, y) \cdot \nabla]c(x, y, t) + S(x, y). \tag{2.7}$$

The spatial difference operators to consider are $D(x, y)\nabla^2$ and $-\boldsymbol{v}(x, y) \cdot \nabla$. The operator $D(x, y)\nabla^2$ is the Laplacian matrix multiplied with the diffusion at the current location and the Laplacian can be approximated by using the nine-point stencil presented in Theorem 2.1. The divergence operator $\boldsymbol{v}(x, y) \cdot \nabla$ is determined through partial derivatives defined in 2.5 as

$$\boldsymbol{v}(x, y) \cdot \nabla = v_x(x, y)\partial_x + v_y(x, y)\partial_y.$$

By using the matrices for ∂_x and ∂_y, it follows that the operator $\boldsymbol{v}(x, y) \cdot \nabla$ can written as a matrix

$$\boldsymbol{v}(x, y) \cdot \nabla = \frac{v_x(x, y)}{2h} \begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} + \frac{v_y(x, y)}{2h} \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}.$$

The final (spatial) difference operator in equation (2.7) at location (x, y) is then defined by the matrix $D(x,y)\nabla^2 - \boldsymbol{v}(x,y)\cdot\nabla$ as

$$D(x,y)\nabla^2 - \boldsymbol{v}(x,y)\cdot\nabla = \begin{bmatrix} \frac{D(x,y)}{4h^2} & \frac{D(x,y)}{2h^2} - \frac{v_y(x,y)}{2h} & \frac{D(x,y)}{4h^2} \\ \frac{D(x,y)}{2h^2} + \frac{v_x(x,y)}{2h} & -3\cdot\frac{D(x,y)}{h^2} & \frac{D(x,y)}{2h^2} - \frac{v_x(x,y)}{2h} \\ \frac{D(x,y)}{4h^2} & \frac{D(x,y)}{2h^2} + \frac{v_y(x,y)}{2h} & \frac{D(x,y)}{4h^2} \end{bmatrix}. \tag{2.8}$$

By considering the above difference matrix, one can see how nearby pixels contribute to the central pixels. The center element describes how the diffusion decreases the concentration from the center and moves it outwards. The face elements describe how the diffusion and the convection together increase/decrease the concentration and finally the corner elements describe how the diffusion from the corners increases the concentration in the central pixel.

It is worth noting that in this implementation, it is possible that some of the face values become negative. This means that the convection tries to move the concentration away from a pixel more than the pixel actually has concentration. Negative concentrations do not represent a physical state and will usually cause a simulation to completely fail. To avoid this, diffusion can be chosen to be just large enough so that all the face points remain positive. That is, at every location (x, y), the conditions

$$\begin{aligned} \frac{D(x,y)}{2h^2} - \frac{|v_x(x,y)|}{2h} \geq 0 \quad &\Leftrightarrow \quad D(x,y) \geq h\,|v_x(x,y)| \quad \text{and} \\ \frac{D(x,y)}{2h^2} - \frac{|v_y(x,y)|}{2h} \geq 0 \quad &\Leftrightarrow \quad D(x,y) \geq h\,|v_y(x,y)|\,, \end{aligned}$$

or equivalently, the condition

$$D(x,y) \geq h\cdot\max\{|v_x(x,y)|\,,|v_y(x,y)|\} \Leftrightarrow h \leq \frac{D(x,y)}{\max\{|v_x(x,y)|\,,|v_y(x,y)|\}}$$

must be satisfied. Instead of the maximum expression, the norm expression $||\boldsymbol{v}(x,y)||_2 = \sqrt{v_x(x,y)^2 + v_y(x,y)^2}$ can be used as

$$D(x,y) \geq h\cdot\sqrt{v_x(x,y)^2 + v_y(x,y)^2} \Leftrightarrow h \leq \frac{D(x,y)}{\sqrt{v_x(x,y)^2 + v_y(x,y)^2}} \tag{2.9}$$

naturally satisfies both conditions. The inequality (2.9) will be referred as a spatial stability condition which sets a limit to the size of discretization h. Unfortunately, even if the diffusion was about the same magnitude as the wind speed, the required discretization would be about the size 1 m or about 10^{-5} in degrees. This discretization is too dense in order to compute in reasonable time and with acceptable memory usage. Instead, the spatial discretization is chosen first, about 100 m or 0.001° at its smallest, and the diffusion scaled accordingly such that the stability condition is satisfied. Depending on the discretization h, the diffusion is chosen to be

$$D(x,y) = C_D\cdot h\cdot\sqrt{v_x(x,y)^2 + v_y(x,y)^2}, \tag{2.10}$$

where C_D is some scaling factor which can be optimized. Typically, the scaling factor at minimum is about one but can be smaller depending on the stability. Although this ends up overestimating the true (turbulent) diffusion, the simulations work well with this choice, as shown in Chapter 5.

The temporal discretization also limits the diffusion. If the diffusion is too large, the concentration may move out of a pixel faster than the concentration increases in that pixel. This is especially important in the source pixel. Initially, the state is zero everywhere and after the first time step, pixels near (or on the top of) the source have positive concentrations. After the second time step, concentration decreases due to diffusion but increases again due to the source term. At the source pixel, this can be written as a process

$$\underset{t=0}{0} \Rightarrow \underset{t=\tau}{\tau Q} \Rightarrow \tau Q + \tau(\underset{t=2\tau}{-3D\tau Q/h^2} + S)$$

where τ is the used time step, D is diffusion at the source location and Q is the emission rate of the source. The term after two time steps must remain non-negative in order to avoid negative concentrations which yields to the requirement

$$2\tau Q - 3D\tau^2 Q/h^2 \geq 0 \Leftrightarrow \tau \leq \frac{2h^2}{3D}$$

which sets an upper limit for the time step. As can been seen, the condition is not depended on the source rate Q. A similar stability condition must be valid at every pixel but the condition is the strictest at the source pixel and at the beginning of the simulation. For other pixels, diffusion and convection from the neighbouring pixels will always increase the concentration in the central pixel more than the diffusion decreases it. Thus, at every location (x, y), a temporal stability condition

$$\tau \leq \frac{2h^2}{3D(x, y)} \tag{2.11}$$

must be satisfied to avoid negative pixels.

The finite difference matrix (2.8) with the stability conditions (2.9) and (2.11) has now been successfully constructed such that instability issues have been avoided. The final term is the source term $S(x, y)$ which can be written as a $N_y \times N_x$ matrix. This matrix is zero everywhere expect near the source. Each point source is assumed to spread to the neighbouring pixels with the weights given by inverse distance weighting presented later in Theorem B.1.

The rate of change in time for the function c is defined by the right-hand side of equation (2.7). The next state can be computed by using the previous state and the forward difference in time. The current state vector $\boldsymbol{c}_t$ is first operated by the spatial difference matrix $\boldsymbol{M}$ and the source term is then added. The update rule for the state vector is

$$\boldsymbol{c}_{t+\tau} = \boldsymbol{c}_t + \tau \cdot (\boldsymbol{M}\boldsymbol{c}_t + S) \tag{2.12}$$

where τ is the time step. The next state is a one-dimensional vector which when reshaped to $N_y \times N_x$ represents concentration values in the $N_y \times N_x$ grid. The difference matrix needs to be updated whenever the wind field (and/or the diffusion) changes.

The grid can and should be very dense for the computations to be as accurate as possible. However, for a dense spatial discretization, the total spatial difference matrix can become very large. For example, if the size of the discretization is about $0.001°$ then difference matrices can have shapes like 500000×500000. Iterating this over each time step by using standard matrix product is extremely slow, not to mention that an object with 500000^2 64-bit floats takes $500000^2 \cdot 64$ bits or $500000^2 \cdot 8 = 2$ Tb (Tera bytes) of (random access) memory which is not manageable in normal circumstances. Luckily, each row of this kind of matrix has only nine non-zero elements so most of the elements in the whole matrix are zeros. For this type of situation, an object called sparse matrix can be used. A sparse matrix only stores elements (and their row and column locations) which are non-zero, saving a lot of memory.

Another way to increase computation speed and reduce memory usage is to use adaptive spatial grids. That is, the spatial grid is only required to cover the area in which the plume exists and is relevant for the observation. A comparison between a normal grid and an adaptive grid is presented in Figure 2.2. Especially for long distances and plumes' with directions like $-135°, -45°, 45°$ or $135°$, the adaptive grid clearly decreases the number of used pixels, which in turn decreases the computation time.

(a) A normal grid. (b) An adaptive grid.

Figure 2.2: A comparison between the normal and adaptive grids. On the left in the normal grid, there are $15 \times 15 = 225$ pixels, and on the right in the adaptive grid, there are $10 \times 18 = 180$ pixels, which is 20 % less pixels in this case.

During the simulation, some of the plume concentration may reach the boundary of the simulation grid in which case it vanishes due to the zero-boundary condition.

The smallest reasonable temporal resolution is 1 second and the smallest reasonable spatial resolution for simulation grids is 0.001 degree. The evaluation becomes faster as the discretization step size becomes larger. Unfortunately, such a simple implementation

of this model using finite differences as used in this thesis may not be accurate enough if either one of the spatial and temporal discretizations is too large. Moreover, the stability conditions (2.9) and (2.11) set a huge limitations for this implementation. Possible improvements, other implementations as well as other existing mass transport models are discussed in Chapter 6.

The required simulation time varies depending on the range and the area in which the plume needs to be simulated. In this thesis, the simulation time is typically from 6 to 8 hours, but may be longer depending on the distance between the source and the observation. Wind fields from external data set have an hourly discretization as will be explained in Chapter 4, and by default are updated every half an hour. At half hour, the wind field is the average of the two closest even hour wind fields.

2.2 Gaussian plume model

A Gaussian plume model is a special case of the general convection-diffusion model (2.2). The fundamental idea behind the Gaussian model is that the plume is assumed to move to the wind direction and spread perpendicularly to the wind direction according to a Gaussian function. A major benefit of the Gaussian model over the convection-diffusion model is significantly faster to evaluate. Downsides of the Gaussian model is that it ignores the altitude aspect of the plume and assumes a homogeneous wind everywhere.

2.2.1 Formulation

The Gaussian plume model used in this thesis is the one presented in Nassar et al. [15] which is a special case of the convection-diffusion equation (2.2). For a single point source can be formulated as

$$V(x,y) = \frac{Q}{\sqrt{2\pi}\sigma(x)u} e^{-\frac{y^2}{2\sigma^2(x)}} \tag{2.13}$$

$$\sigma(x) = a \cdot \left(\frac{x}{x_0}\right)^b \tag{2.14}$$

where V is the amount of CO_2 per atmospheric column (from the ground level to the top of the atmosphere) in g/m^2, $x > 0$ is the downwind direction in m, y is the direction perpendicular to the wind direction in m, Q is the emission rate (or source rate) in g/s, $\sigma(x)$ is the standard deviation representing diffusion in y-direction at distance x, u is the wind speed for the homogeneous wind field inside the considered area in m/s, the atmospheric stability parameter a, b is the exponent to which power the plume vanished when moving away from the source and $x_0 = 1000$ m is the characteristic length which ensures that the argument of the exponent is dimensionless. The expression for $\sigma(x)$ is

determined empirically and can not be derived purely with mathematics. The atmospheric stability parameter is typically between 104 and 213, as explained later, and the exponent b is 0.894 by default, as stated by Nassar et al. in their paper [15].

The parameters which may be fitted for the Gaussian plume are emission rate Q, atmospheric stability parameter a, exponent b and the wind direction. All of the parameters present in the model can not be fitted simultaneously. For example, in the Gaussian model (2.13), the individual values emission Q and the wind speed u do no affect the generated model values as long as the ratio Q/u stays constant. That is, Q and u have a direct linear dependency, as stated also by Nassar et al. [15]. Due to this, it is impossible to find both them together using optimization algorithm. Luckily, the average wind speed u can be obtained from external dataset.

The atmospheric stability parameter a is related to Pasquill stability classes [1]. To put it simply, the stability class is dependent on the surface wind speed and the stability parameter is dependent on the stability class according to Table 2.1. Standard day time Pasquill stability classes are referred with letters A, B and C.

surface wind speed (m/s)	<2	3-5	>5
Pasquill stability class	A	B	C
Atmospheric stability parameter	213	156	104

Table 2.1: Daytime Pasquill stability classes and stability parameter.

The stability parameter changes step-wise when the surface wind speed exceeds a certain threshold. However, as there is no reason for the stability parameter being a step function, an interpolating function for it can also be used as stated by Nassar et al. [16] in which case the points used for interpolation are written in Table 2.2.

surface wind speed (m/s)	2	2.5	4.0	4.5	5.0
Atmospheric stability parameter	213	184.5	156	130	104

Table 2.2: Points and values used for interpolating stability parameter.

Values 184.5 and 130 correspond intermediate stability classes AB and BC [15], correspondingly. If the speed is less than 2 m/s, then the stability is 213 and if the speed larger than 5 m/s, then the stability is 104. Interpolation between speeds 2 m/s and 5 m/s is done by using Shepard's method explained in Appendix B.

The Gaussian model (2.13) can be directly derived from the convection-diffusion equation. The derivation from the general convection-diffusion model to the Gaussian model will be presented next by following guidelines of the derivation introduced by Stockie [22].

First step is to pick a coordinate for the convection-diffusion equation, presented in equation (2.2). There are three spatial dimensions which means $\boldsymbol{x} = (x, y, z)$. A (Cartesian) coordinate system is chosen such that the source is at $(0, 0, h)$ where h is height of the plume source. The x-coordinate is to the direction determined by the first two components of the wind vector $\boldsymbol{v} = (v_x, v_y, v_z)$ at $(0, 0, h)$, the z-coordinate is upwards from the source point and y-coordinate is perpendicular to wind, direction determined by the right-hand rule. Earth's surface is approximated as a plane $z = 0$, ignoring the topography of the surface of earth. In order to derive the expression for the Gaussian plume mode, following assumptions are made.

1. The source term is given by $S(x, y, z) = Q\delta(x)\delta(y)\delta(z - H)$ where δ is the (Dirac) delta function centered at zero, Q is the source rate and $H = h + \Delta h$ is the effective height where h is the height of the actual plume source and Δh is plume rise along x-axis.

2. The convection-diffusion equation is assumed to be in steady state. This means that all variables and parameters are independent of time i.e. $D(\boldsymbol{x}, t) = D(\boldsymbol{x})$, $\boldsymbol{v}(\boldsymbol{x}, t) = \boldsymbol{v}(\boldsymbol{x})$, $c(\boldsymbol{x}, t) = c(\boldsymbol{x})$ and $\partial_t c(\boldsymbol{x}, t) = 0$. All plumes should eventually reach steady state if observation/simulation time period is long enough and both diffusion and wind field remain constant in time and are regular enough.

3. The wind field is assumed to be homogeneous in time and space with some fixed wind speed and a direction is the direction of x-axis. That is, the wind vector is a constant vector $\boldsymbol{v}(\boldsymbol{x}) = (u, 0, 0)$ everywhere where $u > 0$ is the wind speed. The plume is assumed to move only to positive x-direction.

4. Diffusion is assumed to be only dependent on x coordinate. That is, $D(\boldsymbol{x}) = D(x)$.

5. Diffusion in the x-direction is much smaller than convection which means that the term $D(x)\partial_x^2 c(x, y, z)$ is insignificant compared to a convection term $u\partial_x c(x, y, z)$. Moreover, assuming that the diffusion field is regular enough, also the term $\partial_x D(x) \cdot \partial_x c(x, y, z)$ becomes negligible.

With these assumptions, Equation (2.2) becomes

$$u\partial_x c(x, y, z) = D(x)\partial_y^2 c(x, y, z) + D(x)\partial_z^2 c(x, y, z) + Q\delta(x)\delta(y)\delta(z - H) \tag{2.15}$$

where $x \in (0, \infty)$ (plume moves only to the wind direction), $y \in (-\infty, \infty)$ (plume spreads perpendicular to wind direction) and $z \in (0, \infty)$ (plume can not go through the ground). The boundary conditions for this PDE are

$$c(0, y, z) = \lim_{x\to\infty} c(x, y, z) = \lim_{y\to\pm\infty} c(x, y, z) = \lim_{z\to\infty} c(x, y, z) = \partial_z c(x, y, 0) = 0. \tag{2.16}$$

The source term $Q\delta(x)\delta(y)\delta(z-H)$ in Equation (2.15) can be eliminated by introducing the delta function to the boundary conditions [22]. New boundary conditions are then

$$\begin{aligned} c(0,y,z) &= \frac{Q}{u}\delta(y)\delta(z-H) \\ \lim_{x\to\infty} c(x,y,z) &= \lim_{y\to\pm\infty} c(x,y,z) = \lim_{z\to\infty} c(x,y,z) = \partial_z c(x,y,0) = 0 \end{aligned} \tag{2.17}$$

in which case the PDE reduces to

$$u\partial_x c(x,y,z) = D(x)\partial_y^2 c(x,y,z) + D(x)\partial_z^2 c(x,y,z). \tag{2.18}$$

The PDE given in Equation (2.18) will be solved now with boundary conditions (2.17). Define a new variable

$$r = \frac{1}{u}\int_0^x D(s)ds, \tag{2.19}$$

which characterizes the total diffusion (width) of the plume when it is being observed at distance x from the source. By defining $c(x,y,z) = C(r,y,z)$ and applying the chain rule yields

$$u\partial_x c(x,y,z) = u\partial_x C(r,y,z) = u\partial_r C(r,y,z)\cdot\partial_x r = D(x)\partial_r C(r,y,z).$$

The PDE in Equation 2.18 now becomes

$$D(x)\partial_r C(r,y,z) = D(x)\partial_y^2 C(r,y,z) + D(x)\partial_z^2 C(r,y,z)$$

which after dividing with $D(x)$, assuming it is non-zero, yields

$$\partial_r C(r,y,z) = \partial_y^2 C(r,y,z) + \partial_z^2 C(r,y,z). \tag{2.20}$$

New boundary conditions using the variable r are

$$\begin{aligned} c(0,y,z) &= \frac{Q}{u}\delta(y)\delta(z-H), \\ \lim_{r\to\infty} c(r,y,z) &= \lim_{y\to\infty} c(r,y,z) = \lim_{z\to\infty} c(r,y,z) = \partial_z c(r,y,0) = 0. \end{aligned}$$

The PDE in Equation (2.20) can be solved using separation of variables. The solution is assumed to have a form

$$C(r,y,z) = \frac{Q}{u}A(r,y)B(r,z),$$

which after substituting into (2.20) yields

$$\begin{aligned} &\frac{Q}{u}\cdot[B(r,z)\partial_r A(r,y) + A(r,y)\partial_r B(r,z)] = \frac{Q}{u}[B(r,z)\partial_y^2 A(r,y) + A(r,y)\partial_z^2 B(r,z)] \\ \Leftrightarrow\; &B(r,z)\partial_r A(r,y) + A(r,y)\partial_r B(r,z) = B(r,z)\partial_y^2 A(r,y) + A(r,y)\partial_z^2 B(r,z) \end{aligned}$$

To find non-trivial solution for $A(r,y)$ and $B(r,z)$, coefficients for $A(r,y)$ and $B(r,z)$ must be simultaneously equal in both sides. That is, the equation must be true for

all $r \in (0,\infty), y \in (-\infty,\infty), z \in (0,\infty)$ with some non-constant functions $A(r,y)$ and $B(r,z)$. Equating the coefficients yields to two partial differential equations

$$\partial_r A(r,y) = \partial_y^2 A(r,y), A(0,y) = \delta(y), \lim_{r\to\infty} A(r,y) = \lim_{y\to\pm\infty} A(r,y) = 0,$$

$$\partial_r B(r,z) = \partial_z^2 B(r,z), B(0,z) = \delta(z-H), \lim_{r\to\infty} B(r,z) = \lim_{z\to\infty} B(r,z) = \partial_z B(r,0) = 0.$$

The first equation for $A(r,y)$ has a form of one-dimensional heat equation where r acts as the time variable and delta function as the initial condition. The solution is known as the heat kernel [9] and it is given by

$$A(r,y) = \frac{1}{\sqrt{4\pi r}} e^{-\frac{y^2}{4r}}.$$

The second equation for $B(r,y)$ has a similar form although the boundary condition $\partial_z B(r,0) = 0$ with initial condition $B(0,z) = \delta(z-h)$ slightly changes the solution compared to the standard heat kernel. The solution is given by

$$B(r,z) = \frac{1}{\sqrt{4\pi r}} \left(e^{-\frac{(z-H)^2}{4r}} + e^{-\frac{(z+H)^2}{4r}} \right),$$

which satisfies that the boundary conditions for $B(r,z)$ and differentiating verifies that indeed $\partial_r B(r,z) = \partial_z^2 B(r,z)$. The solution for $C(r,y,z)$ is then

$$C(r,y,z) = \frac{Q}{u} A(r,y)B(r,z) = \frac{Q}{4\pi u r} e^{-\frac{y^2}{4r}} \left(e^{-\frac{(z-H)^2}{4r}} + e^{-\frac{(z+H)^2}{4r}} \right). \tag{2.21}$$

The Gaussian plume given by equation (2.21) is shown in Figure 2.3. The plume has Gaussian distribution in both y and z with standard deviations σ_y and σ_z, respectively.

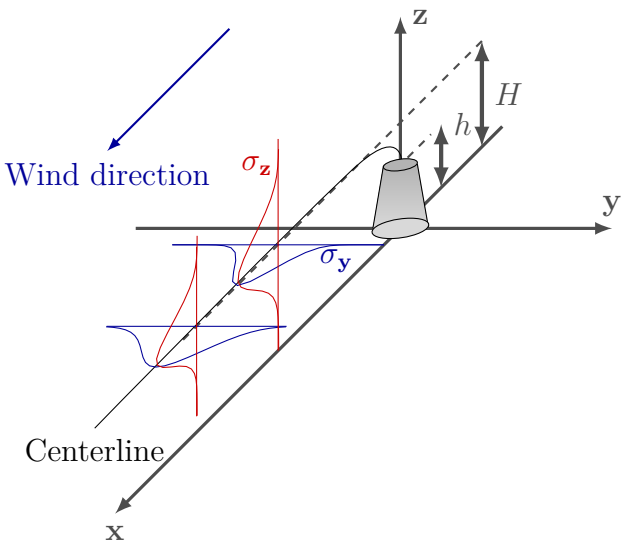


Figure 2.3: A diagram of the Gaussian plume model. The wind direction is marked with a blue arrow in the upper left corner. The red curves indicate the vertical distributions in the z-direction and the blue curves the horizontal distribution in the y-direction. The centerline of the plume is marked with solid black line, and the height to which the plume rises is marked with dashed black line.

The variable r in Equation (2.21) is related to the standard deviation of the plume. Hence, it is natural to define $\sigma^2(x) = 2r$. For a Gaussian plume, the standard deviation has the form $\sigma(x) = ax^b$ where a, b are parameters determined through experimental measurements. Substituting $\sigma^2(x)$ to the solution (2.21) yields to

$$G(x,y,z) = \frac{Q}{2\pi u\sigma^2(x)} e^{-\frac{y^2}{2\sigma^2(x)}} \left(e^{-\frac{(z-H)^2}{2\sigma^2(x)}} + e^{-\frac{(z+H)^2}{2\sigma^2(x)}} \right). \tag{2.22}$$

The equation accounts the height of the plume from the ground level. In reality, a satellite observing the plume is unable to see the actual height of the concentration but rather observes the total vertical column from the ground level to the top of the atmosphere. The concentration per atmospheric column can be computed via integrating the solution (2.22) over the z-axis which yields

$$\begin{aligned}
V(x,y) &= \int_0^\infty G(x,y,z)dz \\
&= \int_0^\infty \frac{Q}{u2\pi\sigma^2(x)} e^{-\frac{y^2}{2\sigma^2(x)}} \left(e^{-\frac{(z-H)^2}{2\sigma^2(x)}} + e^{-\frac{(z+H)^2}{2\sigma^2(x)}} \right) dz \\
&= \frac{Q}{u\sqrt{2\pi}\sigma(x)} e^{-\frac{y^2}{2\sigma^2(x)}} \left(\int_0^\infty \frac{1}{\sqrt{2\pi}\sigma(x)} e^{-\frac{(z-H)^2}{2\sigma^2(x)}} dz + \int_0^\infty \frac{1}{\sqrt{2\pi}\sigma(x)} e^{-\frac{(z+H)^2}{2\sigma^2(x)}} dz \right) \\
&= \frac{Q}{u\sqrt{2\pi}\sigma(x)} e^{-\frac{y^2}{2\sigma^2(x)}} \left(\int_H^\infty \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}z^2} dz + \int_{-H}^\infty \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}z^2} dz \right).
\end{aligned}$$

The function

$$\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}z^2} := \phi(z)$$

is the density function of the standardized normal distribution. Since the normal density is an even function and the integral over the real line is equal to 1, it follows that

$$\int_H^\infty \phi(z)dz + \int_{-H}^\infty \phi(z)dz = \int_H^\infty \phi(z)dz + 1 - \int_{-\infty}^{-H} \phi(z)dz = 1,$$

which after substituting into the equation derived for $V(x,y)$ finally yields

$$V(x,y) = \frac{Q}{u\sqrt{2\pi}\sigma(x)} e^{-\frac{y^2}{2\sigma^2(x)}}, \tag{2.23}$$

which is the same form as the Gaussian plume model given in Equation (2.13) when the standard deviation $\sigma_y(x) = (x/x_0)^{0.894}$.

2.2.2 Numerical implementation

The Gaussian model is simpler and faster to both implement and evaluate compared to the convection-diffusion model as it requires no simulations. Inputting parameters and a location (x, y) gives an estimation on the column average of CO_2. The plume values can

first be generated at pixels in Cartesian xy-grid where x is the direction parallel to the wind and y is direction perpendicular to x. According to the assumptions of the Gaussian plume, the plume exists only when $x > 0$. These coordinates can then be rotated around the source point to the desired wind direction with a rotation matrix

$$\boldsymbol{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}.$$

Applying the matrix to a point $\boldsymbol{x} = (x, y) \in \mathbb{R}^2$ rotates it counterclockwise θ degrees (or radians) around the origin. New coordinates after the rotation are

$$\boldsymbol{Rx} = (x\cos\theta - y\sin\theta, x\sin\theta + y\cos\theta)$$

If the rotation is required around some other point $(x_0, y_0) \neq (0, 0)$, then the rotation can be performed as

$$\boldsymbol{Rx} = ((x - x_0)\cos\theta - (y - y_0)\sin\theta, (x - x_0)\sin\theta + (y - y_0)\cos\theta) + (x_0, y_0).$$

The rotation matrix is unitary, so the inverse is the same as the transpose. That is, the inverse rotation matrix is

$$\boldsymbol{R}^{-1} = \boldsymbol{R}^{\mathrm{T}} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

and applying it to a point $\boldsymbol{x} \in \mathbb{R}^2$ rotates it clockwise by θ degrees. A Gaussian plume to a desired wind direction is generated by rotating points in the longitude-latitude grid around the source point. The grid rotation is illustrated in Figure 2.4. The values remain the same during the rotation, only their corresponding coordinates change.

(a) Before the rotation. (b) After the rotation.

Figure 2.4: A diagram about the grid rotation. As can be observed, the source point, marked with a black dot, stays put while the grid and the red lines representing the plume area are rotated.

The rotated grid is very similar as the adaptive grid, when using the convection-diffusion model for generation. In the case where there are multiple sources, each of them creates their own independent Gaussian plume and each source has a corresponding rotated grid.

The Gaussian plume model requires distances in metres so distances computed in degrees need to be converted to corresponding changes in metres. Due to Earth having a spherical shape, the change of longitude in metres per longitudial degree is dependent on the latitude. At latitude 0° (the Equator), small changes in longitudial and latitudial degree correspond equal small changes in metres along Earth's surface. That is, a change $d\theta$ in longitudial or latitudial degrees corresponds a change

$$ds = \frac{\pi}{180^\circ} \cdot R d\theta$$

along the Earth's surface where R is the radius of Earth ($\approx$ 6371 km). As the latitude increases towards the north pole at 90° or decreases towards the south pole at -90°, the longitudial change in metres per longitudial degree gets smaller. A general law states that a small change in degrees corresponds a change

$$ds = \frac{\pi}{180^\circ} \cdot R \cos(\phi) d\theta$$

on the surface of Earth where ϕ is the latitude. For converting longitudial change from degrees to metres, a latitude of the source point is used.

3. Inverse problems

This chapter covers inverse problems and related theories on solving them. First, classical inverse problems and Hadamard conditions for well-posed inverse problems are introduced. After that, general aspects of solving inverse problems through optimization are discussed. Finally, a concept of inverse crime is discussed briefly.

3.1 Classical inverse problems

Inverse problems are problems in which unknown quantity or parameters are required to be solved using indirect measurements which can be noisy and/or incomplete. A known vector belonging to the data space $\mathbb{R}^m$ is called data vector (or measurement) and it is denoted as $\boldsymbol{y}$. An unknown vector belonging to the state space $\mathbb{R}^n$ is called state vector and it is denoted as $\boldsymbol{x}$. A general mathematical model for an discrete inverse problem using indirect measurements to find out quantity $\boldsymbol{x}$ can be formulated as

$$\boldsymbol{y} = F(\boldsymbol{x}) + \boldsymbol{\epsilon} \tag{3.1}$$

where $F\colon \mathbb{R}^n \to \mathbb{R}^m$ is a forward model and $\boldsymbol{\epsilon} \in \mathbb{R}^m$ is the noise. In the inverse problem, the state vector $\boldsymbol{x}$ is required to be solved given the measurement vector $\boldsymbol{y}$. In comparison, in a direct problem, the vector $\boldsymbol{y}$ is required to be solved given the state vector $\boldsymbol{x}$.

The construction of the forward model F is based on a mathematical, statistical, empirical or physical evidence on how the model should behave. There exists neither explicit formulas nor restrictions for a generic forward model which makes the inverse problems challenging. The noise term $\boldsymbol{\epsilon} \in \mathbb{R}^m$ is something which cannot be modeled using the forward model. This noise can be measurement error, random background noise or modeling error. The noise term can also be multiplicative but in this thesis, it is assumed to be additive.

Different types of problems are characterized via Hadamard conditions [14] first introduced by Jacques Hadamard. According to the Hadamard, a problem can be either well-posed or ill-posed. A well-posed problem is a problem which can usually be solved using linear algebra and satisfies all of the following conditions:

H1: the problem has at least one solution (existence),

H2: the solution is unique (uniqueness), and

H3: the solution depends continuously on data (stability).

If these conditions are met, the forward map F is well-defined, continuous and invertible. If one or more conditions are not met, the problem is ill-posed, leading to an inverse problem.

Well-posed problems are not interesting in this context as they can be solved easily. For example, if the forward map F is linear, in which case it can be written as a $\mathbb{R}^{m\times n}$ matrix, and the problem is well-posed, it can be solved via linear algebra and matrix calculus by using the inverse matrix F^{-1}. For ill-posed problems, this type of naive inversion [14] generally fails. As many problems arisen from the real world applications are ill-posed, the mathematical and statistical methods for solving these inverse problems have been developing rapidly over the past few decades.

Solving an inverse problem is generally done by defining a loss (cost) function which somehow measures the distance between the model and the observed data. After finding a suitable expression for this distance, the problem boils down to an optimization problem. One of the most common ways to measure this distance is to use norms in $\mathbb{R}^n$ and the most common norm is naturally the Euclidean or l^2-norm. The residual (vector) between the model and the data is defined as

$$\boldsymbol{r}(\boldsymbol{x}) = F(\boldsymbol{x}) - \boldsymbol{y}, \tag{3.2}$$

and the norm is

$$||\boldsymbol{r}(\boldsymbol{x})||_2 = ||F(\boldsymbol{x}) - \boldsymbol{y}||_2 = \sqrt{\sum_{i=1} r_i(\boldsymbol{x})^2} = \sqrt{\sum_{i=1}(F_i(\boldsymbol{x}) - y_i)^2}.$$

The benefit of the Euclidean norm is that its square can be written as a matrix product $||\boldsymbol{r}(\boldsymbol{x})||_2^2 = \boldsymbol{r}(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{r}(\boldsymbol{x}) = (F(\boldsymbol{x}) - \boldsymbol{y})^{\mathrm{T}}(F(\boldsymbol{x}) - \boldsymbol{y})$. This is an example of a simple l^2-loss function. The loss function is not unique but can be chosen freely, depending on what properties the solution is required to have.

The first addition to the standard l^2-loss function is to include the covariance matrix $\boldsymbol{\Sigma}_{\text{obs}}$ of the observation values. The observation covariance matrix can be added to the loss function as $\boldsymbol{r}(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}\boldsymbol{r}(\boldsymbol{x}) = (F(\boldsymbol{x})-\boldsymbol{y})^{\mathrm{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}(F(\boldsymbol{x})-\boldsymbol{y})$, which is also known as the weighted l^2-norm of $\boldsymbol{r}(\boldsymbol{x})$. If the covariance matrix is diagonal, each data point has only internal variance and the measurements are uncorrelated of each other. Furthermore, a coefficient $1/2$ is sometimes added in front of the loss function. It does not affect the minimization but its related to the multivariate normal distribution defined in Definition

3.5 and the derivative of the loss function in the proof of Theorem 3.4. The loss function can now be written as

$$\mathcal{L}\colon \mathbb{R}^n \to \mathbb{R}, \mathcal{L}(\boldsymbol{x}) = \frac{1}{2}(F(\boldsymbol{x}) - \boldsymbol{y})^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{obs}}^{-1} (F(\boldsymbol{x}) - \boldsymbol{y}). \tag{3.3}$$

The loss function can also be regularized in order to promote different properties of the state vector $\boldsymbol{x}$. The regularization is done by adding an extra term of the form $(\boldsymbol{x} - \boldsymbol{x}_0)^{\mathrm{T}} \Gamma^{\mathrm{T}} \Gamma (\boldsymbol{x} - \boldsymbol{x}_0)$ to the loss function where Γ is an arbitrary matrix and $\boldsymbol{x}_0$ is an arbitrary vector. This type of regularization is known as the Tiknohov regularization [14]. The choice of Γ is not unique and depends on what qualities of the solution are to be promoted. For example, a common choice is $\Gamma = \sqrt{\alpha} I$ and $\boldsymbol{x}_0 = \boldsymbol{0}$ where α is called a regularization parameter and I is the unit matrix. This kind of choice enforces small norms of $\boldsymbol{x}$ and the regularization parameter α allows a certain control over the regularity of the solution. Using a different kind of choice, e.g. choosing Γ as a discrete difference operator and $\boldsymbol{x}_0 = \boldsymbol{0}$, enforces smoothness of $\boldsymbol{x}$.

If the solution is required to be close to some chosen state $\boldsymbol{x}_0$, this can be enforced by adding a prior information to the loss function in quadratic form $\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_0)^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1} (\boldsymbol{x} - \boldsymbol{x}_0)$ where $\boldsymbol{x}_0$ is a prior value of $\boldsymbol{x}$ and $\boldsymbol{\Sigma}_{\mathrm{pri}}$ is a prior covariance matrix. With the prior information, the loss function becomes

$$\mathcal{L}\colon \mathbb{R}^n \to \mathbb{R}, \mathcal{L}(\boldsymbol{x}) = \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_0)^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1} (\boldsymbol{x} - \boldsymbol{x}_0) + \frac{1}{2}(F(\boldsymbol{x}) - \boldsymbol{y})^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{obs}}^{-1} (F(\boldsymbol{x}) - \boldsymbol{y}). \tag{3.4}$$

That is, the prior information ensures that the optimal $\boldsymbol{x}$ stays near the prior value $\boldsymbol{x}_0$. The prior is said to be flat (no prior information is given) if the inverse prior covariance matrix $\boldsymbol{\Sigma}_{\mathrm{pri}}^{-1}$ is a zero matrix. In this thesis, no prior information has been used.

The goal is to find an optimal vector $\boldsymbol{x}_{\mathrm{opt}}$ which minimizes the chosen loss function or mathematically written as

$$\boldsymbol{x}_{\mathrm{opt}} = \min_{\boldsymbol{x} \in \mathbb{R}^n} \{\mathcal{L}(\boldsymbol{x})\}.$$

Finding the optimal $\boldsymbol{x}_{\mathrm{opt}}$ can be done with various ways. Section 3.3 introduces an easy method for solving the optimal $\boldsymbol{x}$ in the case where the forward map F is linear. In the case of a non-linear forward map, an iterative algorithms, such as Levenberg–Marquardt algorithm explained briefly in Section 3.4, can be used.

By default, a state vector $\boldsymbol{x}$ in the loss function is not bounded during optimization, other than by penalizing the distance between the modeled state and the observed data and the distance to the prior. That is, the optimization is done in whole space $\mathbb{R}^n$. However, some state vectors, such vectors with a negative emission component(s), should be avoided as they do not describe a physical situation. If an invalid state vector is inputted for the model, then the modeled vector $F(\boldsymbol{x})$ is evaluated with invalid values, such as NaN values in Python or Matlab, indicating that the loss function is not minimized. With this, the minimization can be done in some (connected) subset $\Omega \subset \mathbb{R}^n$.

As explained, the key aspect of statistically solving an inverse problem is to compare the distance between the modeled state and the observed data. The modeled state has never a perfect correspondence, no matter how many parameters will be included. In the case of synthetic data, which is data generated by using a model with some noise added to it, the minimization can work "too well". If the generated data is from the same model which is used for modeling the state, then naturally the correspondence between the modeled state and the synthetic data is better than in a general case; this is called an inverse crime. The inverse crime can be avoided by using a different model for defining the loss function than the model used for generating the synthetic data.

3.2 Conditional probability and Bayes' theorem

In order to better understand the minimization of the loss function and what is actually minimized, definition of the conditional probability and Bayes' theorem are formulated briefly. First, a definition of the conditional probability is given in Definition 3.1.

Definition 3.1. Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space, where Ω is a non-empty set, $\mathcal{F}$ is a σ-algebra in the set Ω and $\mathbb{P}\colon \mathcal{F} \to [0,1]$ is a probability measure. The probability of $A \in \mathcal{F}$ under the condition of $B \in \mathcal{F}$ is given by

$$\mathbb{P}(A \,|\, B) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}, \quad \mathbb{P}(B) \neq 0.$$

The Bayes's theorem is related to the conditional probability which tells the probability of an event A given that an event B has been observed. From the Definition 3.1, it follows that $\mathbb{P}(A \cap B) = \mathbb{P}(A \,|\, B)\mathbb{P}(B) = \mathbb{P}(B \,|\, A)\mathbb{P}(A)$. Using this equality, the Bayes' theorem for probabilities states that

$$\mathbb{P}(A \,|\, B) = \frac{\mathbb{P}(B \,|\, A)\mathbb{P}(A)}{\mathbb{P}(B)}, \quad \mathbb{P}(B) \neq 0.$$

The Bayes' theorem can be generalized in the case of countably many events. Consider events $(E)_{i \in I}$ which form a partition of a probability space Ω where I is some index set. The partition means that

$$\Omega = \cup_{i \in I} E_i,$$
$$E_i \cap E_j = \emptyset, \text{ when } i \neq j.$$

The probability of an event A can be written as

$$\begin{aligned} \mathbb{P}(A) &= \mathbb{P}(A \cap \Omega) = \mathbb{P}(A \cap (\cup_{i \in I} E_i)) = \mathbb{P}((\cup_{i \in I}(A \cap E_i))) \quad \big| \text{Pair-wise disjoint sets} \\ &= \sum_{i \in I} \mathbb{P}(A \cap E_i) = \sum_{i \in I} \mathbb{P}(A \,|\, E_i)\mathbb{P}(E_i). \end{aligned}$$

The Bayes' theorem can now be formulated as

$$\mathbb{P}(E_i \,|\, A) = \frac{\mathbb{P}(A \,|\, E_i)\mathbb{P}(E_i)}{\mathbb{P}(A)} = \frac{\mathbb{P}(A \,|\, E_i)\mathbb{P}(E_i)}{\sum_{i \in I} \mathbb{P}(A \,|\, E_i)\mathbb{P}(E_i)},$$

where $\mathbb{P}(E_i \,|\, A)$ is called a posterior probability, $\mathbb{P}(E_i)$ is a prior probability, $\mathbb{P}(A \,|\, E_i)$ is the likelihood probability and $\sum_{i \in I} \mathbb{P}(A \,|\, E_i)\mathbb{P}(E_i)$ is a marginal probability.

Generalizing conditional probability and Bayes' theorem for continuous random variables using density functions is not trivial. The rigorous generalization relies heavily on measure theoretic aspects of random variables. One the most important related theorems is the Radon-Nikodym theorem which allows defining a new probability measure to some subset of a probability space. Random variables from a measure theoretic point of view, Radon-Nikodym theorem as well as rigorous formulation of (regular) conditional probability distribution are introduced in detail in the book by Dudley [5].

Definition 3.2. Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space, where $\Omega \subset \mathbb{R}^n$ is a non-empty set, $\mathcal{F}$ is a σ-algebra in the set Ω and $\boldsymbol{X} \colon \Omega \to \mathbb{R}^n$, $\boldsymbol{Y} \colon \Omega \to \mathbb{R}^m$ are random variables. The conditional probability density function of $\boldsymbol{Y}$ given a realization $\boldsymbol{x}$ of the random variable $\boldsymbol{X}$ is

$$f_{\boldsymbol{Y} \,|\, \boldsymbol{X}}(\boldsymbol{y} \,|\, \boldsymbol{x}) = \frac{f_{\boldsymbol{X},\boldsymbol{Y}}(\boldsymbol{x}, \boldsymbol{y})}{f_{\boldsymbol{X}}(\boldsymbol{x})} = \frac{f_{\boldsymbol{X},\boldsymbol{Y}}(\boldsymbol{x}, \boldsymbol{y})}{\int_{\mathbb{R}^m} f_{\boldsymbol{X},\boldsymbol{Y}}(\boldsymbol{x}, \boldsymbol{y}) d\boldsymbol{y}}, \quad f_{\boldsymbol{X}}(\boldsymbol{x}) > 0$$

where $f_{\boldsymbol{X}}$ is the density of $\boldsymbol{X}$, $f_{\boldsymbol{Y}}$ is the density of $\boldsymbol{Y}$ and $f_{\boldsymbol{X},\boldsymbol{Y}}$ is the joint density.

Definition for the conditional probability density function 3.2 is similar to Definition 3.1 but probabilities have been replaced with integrals of the density functions. For continuous random variables, the conditional density can be formulated as follows.

Similarly as in the case for conditional probability, from the definition of the conditional density 3.2, it follows that $f_{\boldsymbol{X},\boldsymbol{Y}}(\boldsymbol{x}, \boldsymbol{y}) = f_{\boldsymbol{Y} \,|\, \boldsymbol{X}}(\boldsymbol{y} \,|\, \boldsymbol{x}) f_{\boldsymbol{X}}(\boldsymbol{x}) = f_{\boldsymbol{X} \,|\, \boldsymbol{Y}}(\boldsymbol{x} \,|\, \boldsymbol{y}) f_{\boldsymbol{Y}}(\boldsymbol{y})$ which when put together result in the following Bayes' theorem for probability densities

$$f_{\boldsymbol{X} \,|\, \boldsymbol{Y}=\boldsymbol{y}}(\boldsymbol{x}) = \frac{f_{\boldsymbol{Y} \,|\, \boldsymbol{X}=\boldsymbol{x}}(\boldsymbol{y}) f_{\boldsymbol{X}}(\boldsymbol{x})}{f_{\boldsymbol{Y}}(\boldsymbol{y})}.$$

It is common to formulate the Bayes' theorem by using the terms posterior distribution, prior distribution and likelihood function, similarly as they were used in the formulation of the Bayes's theorem for probabilities. The Bayes's theorem acts as the fundamental basis of many probabilistic and statistic applications.

Theorem 3.3. *(Bayes) Consider two random variables* $\boldsymbol{X} \in \mathbb{R}^n$ *and* $\boldsymbol{Y} \in \mathbb{R}^m$. *Assume that a random variable* $\boldsymbol{X}$ *has a prior (distribution) with density* π_{pri} *and a likelihood* $\boldsymbol{Y} \mid \boldsymbol{X}$ *has density* L. *The the posterior (distribution) of* $\boldsymbol{X} \mid \boldsymbol{Y}$ *is given by*

$$\pi_{\text{pos}}(\boldsymbol{x} \mid \boldsymbol{y}) = \frac{L(\boldsymbol{y} \mid \boldsymbol{x})\pi_{\text{pri}}(\boldsymbol{x})}{\int_{\mathbb{R}^n} L(\boldsymbol{y} \mid \boldsymbol{x})\pi_{\text{pri}}(\boldsymbol{x})d\boldsymbol{x}}.$$

Finding an exact form for the posterior is rarely possible. If it is possible, the prior is said to be a conjugate prior for the likelihood function and the posterior distribution has the same distribution family as the prior but with different parameters. For example, when the likelihood is a multivariate normal distribution with a known covariance matrix and mean, then the conjugate prior is also a multivariate normal distribution.

3.3 Linear optimal estimation

An optimal estimation [21] can be used to find a mean vector $\overline{\boldsymbol{x}}$ of $\boldsymbol{x}$ for an inverse problem $\boldsymbol{y} = F(\boldsymbol{x}) + \boldsymbol{\epsilon}$ where F is linear and the loss function is the standard l^2-loss function defined in (3.4). By assuming that the prior and the likelihood have multivariate normal distributions, an analytical distribution for the posterior, which is also a multivariate distribution, can be found. The linearity of F is a very strong requirement and may not true for all parameters of F. However, by using sophisticated guesses or values estimated beforehand for the non-linear parameters of F, the optimal estimation can be used to obtain a quick estimation for the magnitude for all the linear parameter(s) of F. These parameter can then be optimized further with other algorithms, together with non-linear parameters.

The optimal estimation will be derived using two different ways. The first derivation is based on minimization of the loss function using the gradient and the Hessian of the loss function (3.4). Naturally, this method of minimization relies to the fact that the loss function is at least two times continuously differentiable. Generally, the loss function can include terms which are not differentiable, making gradient-based optimizations invalid. In this thesis however, the loss function is always defined through weighted l^2-norms or quadratic forms, which are differentiable functions with respect to $\boldsymbol{x}$.

Theorem 3.4. *Consider a linear inverse problem* $\boldsymbol{y} = F\boldsymbol{x} + \boldsymbol{\epsilon}$ *where* $F \in \mathbb{R}^{m\times n}$ *is a matrix and assume that the loss function* $\mathcal{L}$ *has a quadratic form given by* $\mathcal{L}(\boldsymbol{x}) = \frac{1}{2}(\boldsymbol{x}-\boldsymbol{x}_0)^{\mathrm{T}}\boldsymbol{\Sigma}_{\mathrm{pri}}^{-1}(\boldsymbol{x}-\boldsymbol{x}_0) + \frac{1}{2}(F\boldsymbol{x}-\boldsymbol{y})^{\mathrm{T}}\boldsymbol{\Sigma}_{\mathrm{obs}}^{-1}(F\boldsymbol{x}-\boldsymbol{y})$ *where* $\boldsymbol{x}_0$ *is the prior value,* $\boldsymbol{\Sigma}_{\mathrm{pri}}$ *is the prior covariance matrix and* $\boldsymbol{\Sigma}_{\mathrm{obs}}$ *is the observation covariance matrix. The optimization of* $\boldsymbol{x}$ *is given by*

$$\boldsymbol{x}_{\mathrm{opt}} = (F^{\mathrm{T}}\boldsymbol{\Sigma}_{\mathrm{obs}}^{-1}F + \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1})^{-1}(F^{\mathrm{T}}\boldsymbol{\Sigma}_{\mathrm{obs}}^{-1}\boldsymbol{y} + \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1}\boldsymbol{x}_0)$$

with (posterior) covariance matrix

$$\boldsymbol{\Sigma}_{\mathrm{pos}} = (F^{\mathrm{T}}\boldsymbol{\Sigma}_{\mathrm{obs}}^{-1}F + \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1})^{-1}.$$

If the prior information is not given (flat prior), then in Theorem 3.4 the inverse prior covariance matrix $\boldsymbol{\Sigma}_{\mathrm{pri}}^{-1}$ is a zero matrix. The mean $\boldsymbol{x}_0$ and the covariance matrix $\boldsymbol{\Sigma}_{\mathrm{pri}}$ correspond the prior knowledge of the unknown $\boldsymbol{x}$, the covariance matrix $\boldsymbol{\Sigma}_{\mathrm{obs}}$ includes the noise information, which can include measurement error, background noise or model error, and finally, $\boldsymbol{\Sigma}_{\mathrm{pos}}$ is the posterior covariance matrix.

Proof. By the definition, any covariance matrix $\boldsymbol{\Sigma}$ is symmetrical i.e. $\boldsymbol{\Sigma}^{\mathrm{T}} = \boldsymbol{\Sigma}$. Hence, the gradient of the loss function $\mathcal{L}$ is

$$\nabla\mathcal{L}(\boldsymbol{x}) = \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1}(\boldsymbol{x}-\boldsymbol{x}_0) + F^{\mathrm{T}}\boldsymbol{\Sigma}_{\mathrm{obs}}^{-1}(F\boldsymbol{x}-\boldsymbol{y}) = \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1}\boldsymbol{x} - \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1}\boldsymbol{x}_0 + F^{\mathrm{T}}\boldsymbol{\Sigma}_{\mathrm{obs}}^{-1}F\boldsymbol{x} - F^{\mathrm{T}}\boldsymbol{\Sigma}_{\mathrm{obs}}^{-1}\boldsymbol{y}.$$

Since the loss function included the factor $1/2$, taking the gradient conveniently cancels out this factor. Solving when the gradient is zero vector yields

$$\begin{aligned}
&\boldsymbol{\Sigma}_{\mathrm{pri}}^{-1}\boldsymbol{x} - \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1}\boldsymbol{x}_0 + F^{\mathrm{T}}\boldsymbol{\Sigma}_{\mathrm{obs}}^{-1}F\boldsymbol{x} - F^{\mathrm{T}}\boldsymbol{\Sigma}_{\mathrm{obs}}^{-1}\boldsymbol{y} = \boldsymbol{0}\\
&(F^{\mathrm{T}}\boldsymbol{\Sigma}_{\mathrm{obs}}^{-1}F + \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1})\boldsymbol{x} = F^{\mathrm{T}}\boldsymbol{\Sigma}_{\mathrm{obs}}^{-1}\boldsymbol{y} + \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1}\boldsymbol{x}_0\\
&\boldsymbol{x} = (F^{\mathrm{T}}\boldsymbol{\Sigma}_{\mathrm{obs}}^{-1}F + \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1})^{-1}(F^{\mathrm{T}}\boldsymbol{\Sigma}_{\mathrm{obs}}^{-1}\boldsymbol{y} + \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1}\boldsymbol{x}_0).
\end{aligned}$$

so the optimal value $\boldsymbol{x}_{\mathrm{opt}} = (F^{\mathrm{T}}\boldsymbol{\Sigma}_{\mathrm{obs}}^{-1}F + \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1})^{-1}(F^{\mathrm{T}}\boldsymbol{\Sigma}_{\mathrm{obs}}^{-1}\boldsymbol{y} + \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1}\boldsymbol{x}_0)$.

Next the Hessian of the loss function is required which is mathematically the same as the Jacobian of the gradient. By the linearity of the Jacobian and the fact that the Jacobian of the identity mapping is the identity matrix, the Hessian of the loss function is

$$H_{\mathcal{L}}(\boldsymbol{x}) = \boldsymbol{J}(\nabla\mathcal{L}(\boldsymbol{x})) = \boldsymbol{J}(\boldsymbol{\Sigma}_{\mathrm{pri}}^{-1}\boldsymbol{x} - \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1}\boldsymbol{x}_0 + F^{\mathrm{T}}\boldsymbol{\Sigma}_{\mathrm{obs}}^{-1}F\boldsymbol{x} - F^{\mathrm{T}}\boldsymbol{\Sigma}_{\mathrm{obs}}^{-1}\boldsymbol{y}) = \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1} + F^{\mathrm{T}}\boldsymbol{\Sigma}_{\mathrm{obs}}^{-1}F.$$

The Hessian does not depend on $\boldsymbol{x}$ as taking Hessian of any quadratic form is a constant. The (posterior) covariance matrix $\boldsymbol{\Sigma}_{\mathrm{pos}}$ tells covariance between the estimation $\boldsymbol{x}_{\mathrm{opt}}$ and

the true value $\boldsymbol{x}_{\text{true}}$ and it can be computed as

$$\boldsymbol{\Sigma}_{\text{pos}} = \mathbb{E}[(\boldsymbol{x}_{\text{opt}} - \boldsymbol{x}_{\text{true}})(\boldsymbol{x}_{\text{opt}} - \boldsymbol{x}_{\text{true}})^{\text{T}}]. \tag{3.5}$$

It turns out that this covariance is the same as the inverse Hessian of the loss function or $\boldsymbol{\Sigma}_{\text{pos}} = H_{\mathcal{L}}^{-1}$. This can be shown by starting from the fact that $\nabla\mathcal{L}(\boldsymbol{x}_{\text{opt}}) = 0$ and modifying the expression as

$$\begin{aligned}
0 &= \boldsymbol{\Sigma}_{\text{pri}}^{-1}(\boldsymbol{x}_{\text{opt}} - \boldsymbol{x}_0) + F^{\text{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}(F\boldsymbol{x}_{\text{opt}} - \boldsymbol{y}) \\
&= \boldsymbol{\Sigma}_{\text{pri}}^{-1}(\boldsymbol{x}_{\text{opt}} - \boldsymbol{x}_{\text{true}} + \boldsymbol{x}_{\text{true}} - \boldsymbol{x}_0) + F^{\text{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}(F\boldsymbol{x}_{\text{opt}} - F\boldsymbol{x}_{\text{true}} + F\boldsymbol{x}_{\text{true}} - \boldsymbol{y}) \\
&= \boldsymbol{\Sigma}_{\text{pri}}^{-1}(\boldsymbol{x}_{\text{opt}} - \boldsymbol{x}_{\text{true}}) + \boldsymbol{\Sigma}_{\text{pri}}^{-1}(\boldsymbol{x}_{\text{true}} - \boldsymbol{x}_0) + F^{\text{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}(F\boldsymbol{x}_{\text{opt}} - F\boldsymbol{x}_{\text{true}}) + F^{\text{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}(F\boldsymbol{x}_{\text{true}} - \boldsymbol{y}) \\
&= \boldsymbol{\Sigma}_{\text{pri}}^{-1}(\boldsymbol{x}_{\text{opt}} - \boldsymbol{x}_{\text{true}}) + F^{\text{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}F(\boldsymbol{x}_{\text{opt}} - \boldsymbol{x}_{\text{true}}) - \boldsymbol{\Sigma}_{\text{pri}}^{-1}(\boldsymbol{x}_0 - \boldsymbol{x}_{\text{true}}) - F^{\text{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}(\boldsymbol{y} - F\boldsymbol{x}_{\text{true}}).
\end{aligned}$$

By rearranging the terms yields

$$(\boldsymbol{\Sigma}_{\text{pri}}^{-1} + F^{\text{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}F)(\boldsymbol{x}_{\text{opt}} - \boldsymbol{x}_{\text{true}}) = \boldsymbol{\Sigma}_{\text{pri}}^{-1}(\boldsymbol{x}_0 - \boldsymbol{x}_{\text{true}}) + F^{\text{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}(\boldsymbol{y} - F\boldsymbol{x}_{\text{true}}),$$

which after taking a matrix product between the transpose and the matrix itself implies

$$\begin{aligned}
&(\boldsymbol{\Sigma}_{\text{pri}}^{-1} + F^{\text{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}F)(\boldsymbol{x}_{\text{opt}} - \boldsymbol{x}_{\text{true}})(\boldsymbol{x}_{\text{opt}} - \boldsymbol{x}_{\text{true}})^{\text{T}}(\boldsymbol{\Sigma}_{\text{pri}}^{-1} + F^{\text{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}F)^{\text{T}} \\
=&(\boldsymbol{\Sigma}_{\text{pri}}^{-1}(\boldsymbol{x}_0 - \boldsymbol{x}_{\text{true}}) + F^{\text{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}(\boldsymbol{y} - F\boldsymbol{x}_{\text{true}}))(\boldsymbol{\Sigma}_{\text{pri}}^{-1}(\boldsymbol{x}_0 - \boldsymbol{x}_{\text{true}}) + F^{\text{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}(\boldsymbol{y} - F\boldsymbol{x}_{\text{true}}))^{\text{T}} \\
=&\boldsymbol{\Sigma}_{\text{pri}}^{-1}(\boldsymbol{x}_0 - \boldsymbol{x}_{\text{true}})(\boldsymbol{x}_0 - \boldsymbol{x}_{\text{true}})^{\text{T}}(\boldsymbol{\Sigma}_{\text{pri}}^{-1})^{\text{T}} + F^{\text{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}(\boldsymbol{y} - F\boldsymbol{x}_{\text{true}})(\boldsymbol{y} - F\boldsymbol{x}_{\text{true}})^{\text{T}}(\boldsymbol{\Sigma}_{\text{obs}}^{-1})^{\text{T}}F \\
+&\boldsymbol{\Sigma}_{\text{pri}}^{-1}(\boldsymbol{x}_0 - \boldsymbol{x}_{\text{true}})(\boldsymbol{y} - F\boldsymbol{x}_{\text{true}})^{\text{T}}(\boldsymbol{\Sigma}_{\text{obs}}^{-1})^{\text{T}}F + F^{\text{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}(\boldsymbol{y} - F\boldsymbol{x}_{\text{true}})(\boldsymbol{x}_0 - \boldsymbol{x}_{\text{true}})^{\text{T}}(\boldsymbol{\Sigma}_{\text{pri}}^{-1})^{\text{T}}.
\end{aligned}$$

By taking the expected value from the both sides of this equation, the desired result will follow. Note that due to symmetry of the covariance $(\boldsymbol{\Sigma}_{\text{pri}}^{-1})^{\text{T}} = \boldsymbol{\Sigma}_{\text{pri}}^{-1}$ and $(\boldsymbol{\Sigma}_{\text{obs}}^{-1})^{\text{T}} = \boldsymbol{\Sigma}_{\text{obs}}^{-1}$. This also implies that $(\boldsymbol{\Sigma}_{\text{pri}}^{-1} + F^{\text{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}F)^{\text{T}} = F^{\text{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}F + \boldsymbol{\Sigma}_{\text{pri}}^{-1}$. The expected value of the left side when using the notation 3.5 is becomes

$$\begin{aligned}
&\mathbb{E}[(\boldsymbol{\Sigma}_{\text{pri}}^{-1} + F^{\text{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}F)(\boldsymbol{x}_{\text{opt}} - \boldsymbol{x}_{\text{true}})(\boldsymbol{x}_{\text{opt}} - \boldsymbol{x}_{\text{true}})^{\text{T}}(\boldsymbol{\Sigma}_{\text{pri}}^{-1} + F^{\text{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}F)^{\text{T}}] \\
=&(\boldsymbol{\Sigma}_{\text{pri}}^{-1} + F^{\text{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}F)\mathbb{E}[(\boldsymbol{x}_{\text{opt}} - \boldsymbol{x}_{\text{true}})(\boldsymbol{x}_{\text{opt}} - \boldsymbol{x}_{\text{true}})^{\text{T}}](\boldsymbol{\Sigma}_{\text{pri}}^{-1} + F^{\text{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}F) \\
=&(\boldsymbol{\Sigma}_{\text{pri}}^{-1} + F^{\text{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}F)\boldsymbol{\Sigma}_{\text{pos}}(\boldsymbol{\Sigma}_{\text{pri}}^{-1} + F^{\text{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}F),
\end{aligned}$$

where $\mathbb{E}[(\boldsymbol{x}_{\text{opt}} - \boldsymbol{x}_{\text{true}})(\boldsymbol{x}_{\text{opt}} - \boldsymbol{x}_{\text{true}})^{\text{T}}] = \boldsymbol{\Sigma}_{\text{pos}}$ follows from the fact that the posterior covariance matrix is the covariance between the true state and the estimated state. Similarly, the $\mathbb{E}[(\boldsymbol{x}_0 - \boldsymbol{x}_{\text{true}})(\boldsymbol{x}_0 - \boldsymbol{x}_{\text{true}})^{\text{T}}] = \boldsymbol{\Sigma}_{\text{pri}}$ since the prior covariance matrix is the covariance between the prior state and the true state and $\mathbb{E}[(\boldsymbol{y} - F\boldsymbol{x}_{\text{true}})(\boldsymbol{y} - F\boldsymbol{x}_{\text{true}})^{\text{T}}] = \boldsymbol{\Sigma}_{\text{obs}}$ as the observation covariance matrix is the covariance between the observation and the modeled state. Finally, as the prior state and observation are assumed to be independent, their covariance matrix is a zero matrix or

$$\mathbb{E}[(\boldsymbol{x}_0 - \boldsymbol{x}_{\text{true}})(\boldsymbol{y} - F\boldsymbol{x}_{\text{true}})^{\text{T}}] = \mathbb{E}[(\boldsymbol{y} - F\boldsymbol{x}_{\text{true}})(\boldsymbol{x}_0 - \boldsymbol{x}_{\text{true}})^{\text{T}}] = \boldsymbol{O}.$$

With these, the expected value of the right side is

$$\begin{aligned}&\boldsymbol{\Sigma}_{\text{pri}}^{-1}\boldsymbol{\Sigma}_{\text{pri}}(\boldsymbol{\Sigma}_{\text{pri}}^{-1})^T + F^{\mathrm{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}\boldsymbol{\Sigma}_{\text{obs}}(\boldsymbol{\Sigma}_{\text{obs}}^{-1})^T F + \boldsymbol{\Sigma}_{\text{pri}}^{-1}\boldsymbol{O}(\boldsymbol{\Sigma}_{\text{obs}}^{-1})^{\mathrm{T}}F + F^{\mathrm{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}\boldsymbol{O}(\boldsymbol{\Sigma}_{\text{pri}}^{-1})^{\mathrm{T}}\\ =&\boldsymbol{\Sigma}_{\text{pri}}^{-1} + F^{\mathrm{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}F.\end{aligned}$$

These deductions show that

$$\begin{aligned}&(\boldsymbol{\Sigma}_{\text{pri}}^{-1} + F^{\mathrm{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}F)\boldsymbol{\Sigma}_{\text{pos}}(\boldsymbol{\Sigma}_{\text{pri}}^{-1} + F^{\mathrm{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}F) = \boldsymbol{\Sigma}_{\text{pri}}^{-1} + F^{\mathrm{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}F\\ \Leftrightarrow&\boldsymbol{\Sigma}_{\text{pos}} = (\boldsymbol{\Sigma}_{\text{pri}}^{-1} + F^{\mathrm{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}F)^{-1} = H_{\mathcal{L}}^{-1}\end{aligned}$$

which completes the proof of Theorem 3.4. □

The fact that the posterior covariance matrix is characterized by the (inverse) Hessian is also true in a non-linear case but it depends on the state $\boldsymbol{x}$, unlike in the case of linear model where the covariance matrix is constant everywhere (since Hessian is constant). Given a non-linear model $F\colon \mathbb{R}^n \to \mathbb{R}^m$ and the default l^2-loss function without prior terms $\mathcal{L}(\boldsymbol{x}) = \frac{1}{2}(F(\boldsymbol{x}) - \boldsymbol{y})^{\mathrm{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}(F(\boldsymbol{x}) - \boldsymbol{y})$, the posterior covariance at $\boldsymbol{x}_{\text{opt}}$ is

$$\boldsymbol{\Sigma}_{\text{pos}}(\boldsymbol{x}_{\text{opt}}) = H_{\mathcal{L}}^{-1}(\boldsymbol{x}_{\text{opt}}).$$

where $\boldsymbol{x}_{\text{opt}}$ is the optimal value. This is based on the fact that near $\boldsymbol{x}_{\text{opt}}$ the loss function can be linearized as

$$F(\boldsymbol{x}) \approx F(\boldsymbol{x}_{\text{opt}}) + \frac{\partial F(\boldsymbol{x}_{\text{opt}})}{\partial \boldsymbol{x}}(\boldsymbol{x} - \boldsymbol{x}_{\text{opt}}) = F(\boldsymbol{x}_{\text{opt}}) + \boldsymbol{J}_F(\boldsymbol{x}_{\text{opt}})(\boldsymbol{x} - \boldsymbol{x}_{\text{opt}}) \tag{3.6}$$

where J_F is the Jacobian matrix of the forward map. This is analogical to the one-dimensional case where a differentiable function $f\colon \mathbb{R} \to \mathbb{R}$ can be approximated near x_0 as

$$f(x) \approx f(x_0) + f'(x_0) \cdot (x - x_0).$$

This approximation and the multidimensional formulation in 3.6 are first order Taylor series approximations of the functions f and F near fixed points x_0 and $\boldsymbol{x}_{\text{opt}}$, respectively. Interestingly, the Hessian of the loss function can be characterized by the Jacobian of the forward map. By applying the first order approximation for the forward map F, the standard l^2-loss function can be approximated near $\boldsymbol{x}_{\text{opt}}$ as

$$\mathcal{L}(\boldsymbol{x}) \approx \frac{1}{2}(F(\boldsymbol{x}_{\text{opt}}) + \boldsymbol{J}_F(\boldsymbol{x}_{\text{opt}})(\boldsymbol{x} - \boldsymbol{x}_{\text{opt}}) - \boldsymbol{y})^{\mathrm{T}}\boldsymbol{\Sigma}_{\text{obs}}^{-1}(F(\boldsymbol{x}_{\text{opt}}) + \boldsymbol{J}_F(\boldsymbol{x}_{\text{opt}})(\boldsymbol{x} - \boldsymbol{x}_{\text{opt}}) - \boldsymbol{y})$$

which has Hessian

$$H_{\mathcal{L}} = J_F(\boldsymbol{x}_{\text{opt}})^T\boldsymbol{\Sigma}_{\text{obs}}^{-1}J_F(\boldsymbol{x}_{\text{opt}}).$$

With the prior information included, the Hessian would naturally be

$$H_{\mathcal{L}} = J_F(\boldsymbol{x}_{\text{opt}})^T\boldsymbol{\Sigma}_{\text{obs}}^{-1}J_F(\boldsymbol{x}_{\text{opt}}) + \boldsymbol{\Sigma}_{\text{pri}}^{-1}.$$

The covariance matrix at $\boldsymbol{x}_{\text{opt}}$ is the inverse of the Hessian at $\boldsymbol{x}_{\text{opt}}$ as stated earlier and thus, the covariance can be estimated using the approximated Hessian as

$$\boldsymbol{\Sigma}_{\text{pos}}(\boldsymbol{x}_{\text{opt}}) = (H_{\mathcal{L}})^{-1} \approx (J_F(\boldsymbol{x}_{\text{opt}})^T \boldsymbol{\Sigma}_{\text{obs}}^{-1} J_F(\boldsymbol{x}_{\text{opt}}) + \boldsymbol{\Sigma}_{\text{pri}}^{-1})^{-1} \tag{3.7}$$

Note that this is analogical to the result of Theorem 3.4 but the linear model matrix F has been replaced with the Jacobian matrix. The formula (3.7) gives an useful method for computing covariance matrix at $\boldsymbol{x}_{\text{opt}}$ and quantify error estimations for the retrieved parameters. The linearization of the loss function near some point $\boldsymbol{x}$ is related to the iteration done by the Levenberg–Marquardt (LM) algorithm given in Section 3.4.

Alternative deduction to validate the optimal estimation can be done by using multivariate normal distributions.

Definition 3.5. If a random variable $\boldsymbol{Z}$ taking values in $\mathbb{R}^n$ follows a multivariate normal distribution, it has a density function

$$f_{\boldsymbol{Z}}(\boldsymbol{z}) = \frac{1}{\sqrt{(2\pi)^n \, |\boldsymbol{\Sigma}|}} e^{-\frac{1}{2}(\boldsymbol{z}-\boldsymbol{\mu})^{\mathrm{T}} \boldsymbol{\Sigma}^{-1} (\boldsymbol{z}-\boldsymbol{\mu})}$$

where $\boldsymbol{\mu}$ is the mean of $\boldsymbol{Z}$, $\boldsymbol{\Sigma}$ is a positive-definite covariance matrix of $\boldsymbol{Z}$ and $|\boldsymbol{\Sigma}| = \det(\boldsymbol{\Sigma})$. A random variable $\boldsymbol{Z}$ following a normal distribution is denoted as $\boldsymbol{Z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

By assuming that the forward model F is linear and that both the prior knowledge of $\boldsymbol{x}$ and background noise $\boldsymbol{\epsilon}$ follow the multivariate normal distribution, a similar result as in Theorem 3.4 can be recovered.

Theorem 3.6. *Assume that distributions of $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{\epsilon} \in \mathbb{R}^m$ are given by $\mathcal{N}(\boldsymbol{x}_0, \boldsymbol{\Sigma}_{\text{pri}})$ and $\mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}_{\text{obs}})$ correspondingly. Consider an inverse problem $\boldsymbol{y} = F\boldsymbol{x} + \boldsymbol{\epsilon}$ where $F \in \mathbb{R}^{m \times n}$ is a matrix. The posterior distribution of $\boldsymbol{x}$ given observation $\boldsymbol{y}$ has a form $\pi_{\text{pos}}(\boldsymbol{x} \,|\, \boldsymbol{y}) \propto \exp(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_{\text{opt}})^{\mathrm{T}} \boldsymbol{\Sigma}_{\text{pos}}^{-1} (\boldsymbol{x} - \boldsymbol{x}_{\text{opt}}))$ where*

$$\boldsymbol{x}_{\text{opt}} = (F^{\mathrm{T}} \boldsymbol{\Sigma}_{\text{obs}}^{-1} F + \boldsymbol{\Sigma}_{\text{pri}}^{-1})^{-1} (F^{\mathrm{T}} \boldsymbol{\Sigma}_{\text{obs}}^{-1} \boldsymbol{y} + \boldsymbol{\Sigma}_{\text{pri}}^{-1} \boldsymbol{x}_0),$$

with the posterior covariance matrix

$$\boldsymbol{\Sigma}_{\text{pos}} = (F^{\mathrm{T}} \boldsymbol{\Sigma}_{\text{obs}}^{-1} F + \boldsymbol{\Sigma}_{\text{pri}}^{-1})^{-1}.$$

Again, if the prior is flat, then the inverse covariance matrix of the prior $\boldsymbol{\Sigma}_{\text{pri}}^{-1}$ is a zero matrix. Theorem 3.6 essentially states the same result as Theorem 3.4 but with slightly different approach.

Proof. Since $\boldsymbol{y} = F\boldsymbol{x} + \boldsymbol{\epsilon}$, it implies that $\boldsymbol{\epsilon} = F\boldsymbol{x} - \boldsymbol{y}$ which means that the density of the noise $\boldsymbol{\epsilon}$ is the same as the likelihood of $\boldsymbol{Y} \mid \boldsymbol{X}$. By using the normal density given in Definition 3.5 and denoting the density of the likelihood $\boldsymbol{Y} \mid \boldsymbol{X}$ with $L(\boldsymbol{y} \mid \boldsymbol{x})$, it follows that

$$-2 \ln L(\boldsymbol{y} \mid \boldsymbol{x}) = \boldsymbol{\epsilon}^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{obs}}^{-1} \boldsymbol{\epsilon} + c_1 = (F\boldsymbol{x} - \boldsymbol{y})^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{obs}}^{-1} (F\boldsymbol{x} - \boldsymbol{y}) + c_1,$$

where c_1 is a constant resulted by the normalizing coefficient in front of the exponent after using the logarithm rules. Note that the negative logarithm of the likelihood $-\ln L(\boldsymbol{y} \mid \boldsymbol{x})$ includes a term which is exactly the same as the loss function $\mathcal{L}(\boldsymbol{x})$ given in Equation 3.3. That is, the likelihood $L(\boldsymbol{y} \mid \boldsymbol{x}) \propto \exp(-\mathcal{L}(\boldsymbol{x}))$.

Similarly for the prior

$$-2 \ln \pi_{\mathrm{pri}}(\boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{x}_0)^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1} (\boldsymbol{x} - \boldsymbol{x}_0) + c_2,$$

where again c_2 is some constant. By using Bayes' theorem 3.3, it follows that

$$\pi_{\mathrm{pos}}(\boldsymbol{x} \mid \boldsymbol{y}) \propto \pi_{\mathrm{pri}}(\boldsymbol{x}) L(\boldsymbol{y} \mid \boldsymbol{x})$$

and for the logarithm

$$-2 \ln L(\boldsymbol{x} \mid \boldsymbol{y}) = (\boldsymbol{x} - \boldsymbol{x}_0)^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1} (\boldsymbol{x} - \boldsymbol{x}_0) + (F\boldsymbol{x} - \boldsymbol{y})^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{obs}}^{-1} (F\boldsymbol{x} - \boldsymbol{y}) + c_3,$$

where c_3 contains all constant terms. The posterior density $\pi_{\mathrm{pos}}(\boldsymbol{x} \mid \boldsymbol{y})$ is required to have a form

$$-2 \ln \pi_{\mathrm{pos}}(\boldsymbol{x} \mid \boldsymbol{y}) = (\boldsymbol{x} - \boldsymbol{x}_{\mathrm{opt}})^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{pos}}^{-1} (\boldsymbol{x} - \boldsymbol{x}_{\mathrm{opt}}) + c_4.$$

That is,

$$(\boldsymbol{x} - \boldsymbol{x}_0)^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1} (\boldsymbol{x} - \boldsymbol{x}_0) + (F\boldsymbol{x} - \boldsymbol{y})^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{obs}}^{-1} (F\boldsymbol{x} - \boldsymbol{y}) + c_3 = (\boldsymbol{x} - \boldsymbol{x}_{\mathrm{opt}})^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{pos}}^{-1} (\boldsymbol{x} - \boldsymbol{x}_{\mathrm{opt}}) + c_4$$

must be true $\forall \boldsymbol{x} \in \mathbb{R}^n$. After opening parentheses, this yields to an equality

$$\begin{aligned}
&\boldsymbol{x}^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1} \boldsymbol{x} + \boldsymbol{x}_0^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1} \boldsymbol{x}_0 - \boldsymbol{x}^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1} \boldsymbol{x}_0 - \boldsymbol{x}_0^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1} \boldsymbol{x} + c_2 \\
+&\boldsymbol{y}^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{obs}}^{-1} \boldsymbol{y} + \boldsymbol{x}^{\mathrm{T}} F^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{obs}}^{-1} F \boldsymbol{x} - \boldsymbol{y}^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{obs}}^{-1} F \boldsymbol{x} - \boldsymbol{x}^{\mathrm{T}} F^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{obs}}^{-1} \boldsymbol{y} + c_3 \\
=&\boldsymbol{x}^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{pos}}^{-1} \boldsymbol{x} + \boldsymbol{x}_{\mathrm{opt}}^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{pos}}^{-1} \boldsymbol{x}_{\mathrm{opt}} - \boldsymbol{x}^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{pos}}^{-1} \boldsymbol{x}_{\mathrm{opt}} - \boldsymbol{x}_{\mathrm{opt}}^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{pos}}^{-1} \boldsymbol{x} + c_4.
\end{aligned}$$

Since this must be true for all $\boldsymbol{x} \in \mathbb{R}^n$, the coefficients of terms with both $\boldsymbol{x}^{\mathrm{T}}$ and $\boldsymbol{x}$, terms with $\boldsymbol{x}^{\mathrm{T}}$ and terms with $\boldsymbol{x}$ must be equal.
Equating terms with both $\boldsymbol{x}^{\mathrm{T}}$ and $\boldsymbol{x}$ yields

$$\begin{aligned}
&\boldsymbol{x}^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1} \boldsymbol{x} + \boldsymbol{x}^{\mathrm{T}} F^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{obs}}^{-1} F \boldsymbol{x} = \boldsymbol{x}^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{pos}}^{-1} \boldsymbol{x} \quad \forall \boldsymbol{x} \in \mathbb{R}^n \\
\Leftrightarrow &\boldsymbol{\Sigma}_{\mathrm{pos}}^{-1} = F^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{obs}}^{-1} F + \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1} \\
\Leftrightarrow &\boldsymbol{\Sigma}_{\mathrm{pos}} = (F^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{obs}}^{-1} F + \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1})^{-1}.
\end{aligned} \tag{3.8}$$

and equating terms with $\boldsymbol{x}^{\mathrm{T}}$ yields

$$\begin{aligned}
&-\boldsymbol{x}^{\mathrm{T}}\boldsymbol{\Sigma}_{\mathrm{pri}}^{-1}\boldsymbol{x}_0 - \boldsymbol{x}^{\mathrm{T}}F^{\mathrm{T}}\boldsymbol{\Sigma}_{\mathrm{obs}}^{-1}\boldsymbol{y} = -\boldsymbol{x}^{\mathrm{T}}\boldsymbol{\Sigma}_{\mathrm{pos}}^{-1}\boldsymbol{x}_{\mathrm{opt}} \quad \forall \boldsymbol{x} \in \mathbb{R}^n \\
\Leftrightarrow &\boldsymbol{\Sigma}_{\mathrm{pos}}^{-1}\boldsymbol{x}_{\mathrm{opt}} = \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1}\boldsymbol{x}_0 + F^{\mathrm{T}}\boldsymbol{\Sigma}_{\mathrm{obs}}^{-1}\boldsymbol{y} \\
\Leftrightarrow &\boldsymbol{x}_{\mathrm{opt}} = \boldsymbol{\Sigma}_{\mathrm{pos}}(\boldsymbol{\Sigma}_{\mathrm{pri}}^{-1}\boldsymbol{x}_0 + F^{\mathrm{T}}\boldsymbol{\Sigma}_{\mathrm{obs}}^{-1}\boldsymbol{y}) \quad \big|\text{Substitute } \boldsymbol{\Sigma}_{\mathrm{pos}} \\
\Leftrightarrow &\boldsymbol{x}_{\mathrm{opt}} = (F^{\mathrm{T}}\boldsymbol{\Sigma}_{\mathrm{obs}}^{-1}F + \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1})^{-1}(F^{\mathrm{T}}\boldsymbol{\Sigma}_{\mathrm{obs}}^{-1}\boldsymbol{y} + \boldsymbol{\Sigma}_{\mathrm{pri}}^{-1}\boldsymbol{x}_0), \qquad (3.9)
\end{aligned}$$

which is already enough to prove Theorem 3.6. Equating terms with $\boldsymbol{x}$ or constants terms does not provide any additional information. □

The optimal estimation is equivalent to the posterior mode estimation when the posterior is of the form $\pi_{\mathrm{pos}} \propto e^{-\mathcal{L}}$. Minimizing the loss function $\mathcal{L}$ trivially maximizes the function $e^{-\mathcal{L}}$. This type of choice is very typical for the posterior distribution.

3.4 Levenberg–Marquardt algorithm

Usually, the forward model F is not linear, or is linear only in some variables. For a non-linear optimization with the standard l^2-loss function without prior knowledge or observation covariance, the Levenberg–Marquardt algorithm proceeds as given in Definition 3.7. There are also other possible algorithms for optimizing a non-linear least-square fitting but in this thesis, the LM-algorithm will be used.

Definition 3.7. Given a parameter vector $\boldsymbol{x}$, an observed data $\boldsymbol{y}$ and a residual vector $\boldsymbol{r}(\boldsymbol{x}) = F(\boldsymbol{x}) - \boldsymbol{y}$. To find the minimum of $\mathcal{L}(\boldsymbol{x}) = \boldsymbol{r}(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{r}(\boldsymbol{x}) = ||F(\boldsymbol{x}) - \boldsymbol{y}||_2^2$, a chosen initial value $\boldsymbol{x}_0$ is updated according to the rule

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + (J_F(\boldsymbol{x}_i)^{\mathrm{T}}J_F(\boldsymbol{x}_i) + \lambda_i \mathrm{diag}(J_F(\boldsymbol{x}_i)^{\mathrm{T}}J_F(\boldsymbol{x}_i)))^{-1}J_F(\boldsymbol{x}_i)^{\mathrm{T}}\boldsymbol{r}(\boldsymbol{x_i}),$$

where $J_F(\boldsymbol{x}_i)$ is the Jacobian matrix of F at $\boldsymbol{x}_i$ and the parameter λ_i is a parameter which is used for defining a termination condition for the iteration.

Details and implementation of the Levenberg–Marquardt algorithm are explained in the book "Numerical Recipes" [19]. This thesis uses implementation from Python's SciPy [24] optimization module. The LM-algorithm usually gives very good results for the retrieved parameters, as will be shown by the results in Chapter 5. The covariance matrix for the retrieved parameters can be estimated by using the numerical Jacobian of the forward model at the optimal state as stated in the approximation (3.7) for the covariance matrix. The Jacobian at the optimal state is computed by the LM-algorithm during the final iteration which can be returned; external codes for computing the Jacobian are not required.

3.5 Statistical inversion

Statistical inversion methods are based on random sampling. The main idea is to sample from a distribution of some unknown variable without actually knowing the exact form of the distribution. As the sample size increases, the central limit theorem [6] guarantees that the sample mean converges to the true expected value of the distribution.

3.5.1 Markov chain Monte Carlo method

The term "Monte Carlo" originally refers to Monte Carlo integration which simply means integration by using random sampling. As an example, consider a function $f\colon [a,b] \to \mathbb{R}$ which is assumed to be continuous and non-negative for simplicity. A one-dimensional integral $\int_a^b f(x)dx$ is required to be approximated. The integral has an upper bound

$$\int_a^b f(x)dx \leq (b-a) \cdot \sup_{x\in[a,b]} f(x) = (b-a) \cdot ||f||_\infty .$$

which geometrically represents an area of a rectangle with width $b-a$ and height $||f||_\infty$. Random numbers are then sampled from this rectangle such that values in the x-axis are sampled from $[a,b]$ and values from the y-axis are sampled from $[0, ||f||_\infty]$ using uniform distributions. Each point is accepted or rejected, based on whether or not it falls under the curve of f. This is illustrated in Figure 3.1.

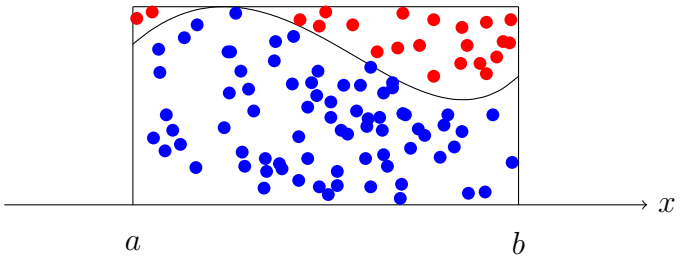


Figure 3.1: An example of the Monte Carlo integration. The function f, marked with solid black line, is the decision boundary. Out of the generated points, the accepted points are underneath the boundary in blue and the rejected points are above in red.

As number of samples increases, the ratio between the number of accepted points and the total number of samples converges to the true value of the integral. That is,

$$\frac{N_{\text{accepted}}}{N_{\text{accepted}} + N_{\text{rejected}}} \to \int_a^b f(x)dx$$

as the sample size increases.

Monte Carlo integration works also for multidimensional integrals over compact sets though the number of required samples grows to the power of the dimension. By

using Monte Carlo method, many kinds of quantities defined through integral, such as an expected value, can be estimated. As an example, assume that a function $f\colon \mathbb{R}^n \to \mathbb{R}$ is a measurable function and π is a density function of a random variable $\boldsymbol{X}$. The expected value is defined as

$$\mathbb{E}[f(\boldsymbol{X})] = \int_{\mathbb{R}^n} f(\boldsymbol{x})\pi(\boldsymbol{x})d\boldsymbol{x}.$$

If the expected value exists, by the strong law of large numbers [6] for a random sample $\boldsymbol{x}_i$ with sample size N from the distribution of $\boldsymbol{X}$ holds

$$\frac{1}{N}\sum_{i=1}^{N} f(\boldsymbol{x}_i) \underset{N\to\infty}{\to} \mathbb{E}[f(\boldsymbol{x})]$$

almost surely. That is, the expected value $\mathbb{E}[f(\boldsymbol{x})]$ can be estimated simply by drawing enough samples from the distribution of $\boldsymbol{X}$, mapping them with f and finally averaging those values. The estimation is then

$$\frac{1}{N}\sum_{i=1}^{N} f(\boldsymbol{x}_i) \approx \mathbb{E}[f(\boldsymbol{x})]$$

for large enough N.

The term "Markov chain" refers to a stochastic process or a sequence of random variables which have a Markov property. In simple terms, this means that the next state depends only the present state [6].

Definition 3.8. A sequence $(X_n)_{n\in\mathbb{N}}$ of discrete random variables is a Markov chain if it satisfies a Markov property

$$\mathbb{P}(X_{n+1} = x \,|\, X_0 = x_0, \ldots, X_n = x_n) = \mathbb{P}(X_{n+1} = x \,|\, X_n = x_n).$$

The sequence of random variables moves from the current state to the next state with transition probability.

Definition 3.9. Let $(S, \mathcal{F})$ be a measurable and countable state space for each X_n. A function $p\colon \mathcal{F} \times S \to [0, 1]$ is called a transition probability (kernel) if

1. For each $s \in S$ the function $A \to p(A, s)$ is a probability measure on $(S, \mathcal{F})$,
2. For each $A \in \mathcal{F}$ the function $s \to p(A, s)$ is a $\mathcal{F}$-measurable.

If the state space is uncountable (like $\mathbb{R}^n$), Markov chain is called Harris chain [6].

Definition 3.10. Let $(S, \mathcal{F})$ be a measurable state space for each X_n and a transition probability $p(x, C) = \mathbb{P}(X_{n+1} \in C \,|\, X_n = x)$ for all states $x \in S$ and measurable sets $C \in \mathcal{F}$. A sequence $(X_n)_{n\in\mathbb{N}}$ is a Harris chain if there exists a set $A \subset S$, an arbitrary $\epsilon > 0$ and a probability measure μ on S such that

1. $\mathbb{P}(\inf\{n \geq 0 \colon X_n \in A\} < \infty \,|\, X_0 = x_0) = 1$ for all initial states $x_0 \in S$,
2. If $x \in A$ is a state and $C \subset S$ is a measurable set, then $p(x, C) \geq \epsilon\mu(C)$.

The expression $\inf\{n \geq 0 \colon X_n \in A\}$ represents the first time when the random process enters the region A and by requiring $\inf\{n \geq 0 \colon X_n \in A\} < \infty$ ensures that the set A is reachable by the random process. The probability

$$\mathbb{P}(\inf\{n \geq 0 \colon X_n \in A\} < \infty \,|\, X_0 = x_0) = 1$$

ensures that the process returns to state inside A almost surely regardless of the initial state. The condition $p(x, C) \geq \epsilon\mu(C)$ is related to the process moving from the state x to some state inside C. The next state $x_{n+1} \in S$ is generated by flipping a biased coin with success probability ϵ. If the coin flip is successful or if $\epsilon \geq 1$, the next state is determined by the probability measure μ and if the coin flip fails, the next state is chosen using the probability $p(x, C)$. This is the same as random walking in the chosen state space S.

After the random walk has been iterated long enough, the distribution of random variables in the chain should converge to some stationary distribution. That is, the values generated from the random walking follow a fixed distribution with some mean and standard deviation. Thus, generating enough values in the Markov chain enables the possibility to sample directly from the stationary distribution without knowing the exact form for it. Using a similar kind of idea, when the prior and the likelihood are known, values from the posterior in Theorem 3.3 can be generated using random walk without actually knowing the posterior distribution; the posterior acts as the stationary distribution. This type of method combining random walk and sampling is called a Markov chain Monte Carlo. There exists many numerical algorithms to sample long chains of values from a posterior distribution; perhaps one of the most known of them is called Metropolis-Hastings algorithm. In this thesis, an adaptive Metropolis algorithm [11] will be used for generating these random walk chains.

3.5.2 Adaptive Metropolis algorithm

Metropolis-Hastings (MH) algorithm is one of the most used implementations for numerically computing MCMC method. Consider a random variable $\boldsymbol{X}$ which has a target

distribution supported in some $S \subset \mathbb{R}^n$ with density p. A standard Metropolis-Hastings algorithm progresses as follows:

1. Define a proposal distribution of $\boldsymbol{Z} \mid \boldsymbol{X}_{t-1}$ with a density q and sample a candidate point $\boldsymbol{z}$ from this proposal distribution,

2. Define an acceptance ratio (probability) as

$$A = \begin{cases} \min\left(\frac{p(\boldsymbol{z})q(\boldsymbol{z} \mid \boldsymbol{x}_{t-1})}{p(\boldsymbol{x}_{t-1})q(\boldsymbol{x}_{t-1} \mid \boldsymbol{z})}, 1\right), \text{ if } p(\boldsymbol{x}_{t-1})q(\boldsymbol{x}_{t-1} \mid \boldsymbol{z}) > 0 \\ 1, \text{ if } p(\boldsymbol{x}_{t-1})q(\boldsymbol{x}_{t-1} \mid \boldsymbol{z}) = 0. \end{cases}$$

   ,

3. Sample a value u from the uniform distribution $\mathcal{U}([0,1])$. If $u \leq A$, the proposed point is accepted and the next value in the chain is $\boldsymbol{x}_t = \boldsymbol{z}$. Otherwise, the point is rejected and the next value in the chain is $\boldsymbol{x}_t = \boldsymbol{x}_{t-1}$.

The longer the algorithm iterates, the more certain it is that the Markov chain has reached the stationary distribution. There are no explicit rules to tell when the convergence has been reached as it is dependent on initial condition, proposal distribution and other parameters. It is also worth noting that the next value can not jump outside the specified set S as the density p vanishes outside S, making the acceptance ratio zero.

The adaptive Metropolis (AM) algorithm [11] slightly deviates from the standard MH algorithm. The term "adaptive" refers to the fact that the proposal distribution with density $q = q_t$ will be updated during the iteration of the algorithm. A proposal distribution is fixed to be a multivariate Gaussian distribution with mean $\boldsymbol{x}_{t-1}$ and covariance $C_t = C_t(\boldsymbol{X}_0, \ldots, \boldsymbol{X}_{t-1})$. It is worth noting that using this kind of covariance makes the chain non-Markovian as the next value now depends on the entire history in the chain. As the proposal distribution is a multivariate Gaussian, the symmetry implies

$$\begin{aligned} q_t(\boldsymbol{z} \mid \boldsymbol{x}_0, \ldots, \boldsymbol{x}_{t-1}) &= \frac{1}{\sqrt{(2\pi)^n |C_t|}} e^{-\frac{1}{2}(\boldsymbol{z}-\boldsymbol{x}_{t-1})^{\mathrm{T}} C_t^{-1} (\boldsymbol{z}-\boldsymbol{x}_{t-1})} \\ &= \frac{1}{\sqrt{(2\pi)^n |C_t|}} e^{-\frac{1}{2}(\boldsymbol{x}_{t-1}-\boldsymbol{z})^{\mathrm{T}} C_t^{-1} (\boldsymbol{x}_{t-1}-\boldsymbol{z})} \\ &= q_n(\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{t-1} \mid \boldsymbol{z}) \end{aligned}$$

in which case the acceptance ratio becomes

$$A = \begin{cases} \min\left(\frac{p(\boldsymbol{z})}{p(\boldsymbol{x}_{t-1})}, 1\right), \text{ if } p(\boldsymbol{x}_{t-1}) > 0 \\ 1, \text{ if } p(\boldsymbol{x}_{t-1}) = 0. \end{cases}$$

The target distribution is generally the posterior of $\boldsymbol{X} \mid \boldsymbol{Y}$ with density π_{pos} where $\boldsymbol{X}$ is the random variable of interest as $\boldsymbol{Y}$ characterizes e.g. a measured observation. According to the Bayes's theorem, for the posterior

$$p(\boldsymbol{x}) = \pi_{\text{pos}}(\boldsymbol{x} \mid \boldsymbol{y}) \propto L(\boldsymbol{y} \mid \boldsymbol{x})\pi_{\text{pri}}(\boldsymbol{x}),$$

where L is the likelihood of $\boldsymbol{Y} \mid \boldsymbol{X}$ and $\pi_{\boldsymbol{X}} = \pi_{\text{pri}}$ is the prior of $\boldsymbol{X}$. The acceptance ratio is then

$$A = \begin{cases} \min\left(\frac{L(\boldsymbol{y} \mid \boldsymbol{z})\pi_{\text{pri}}(\boldsymbol{z})}{L(\boldsymbol{y} \mid \boldsymbol{x}_{t-1})\pi_{\text{pri}}(\boldsymbol{x}_{t-1})}, 1\right), \text{ if } L(\boldsymbol{y} \mid \boldsymbol{x}_{t-1})\pi_{\text{pri}}(\boldsymbol{x}_{t-1}) > 0, \\ 1, \text{ if } L(\boldsymbol{y} \mid \boldsymbol{x}_{t-1})\pi_{\text{pri}}(\boldsymbol{x}_{t-1}) = 0. \end{cases}$$

In the case of a flat prior, the ratio between the priors is always constant in S. Moreover, the likelihood $L(\boldsymbol{y} \mid \boldsymbol{x})$ and the loss function $\mathcal{L}(\boldsymbol{x})$ are (usually) related by

$$\mathcal{L}(\boldsymbol{x}) = -\ln L(\boldsymbol{y} \mid \boldsymbol{x}) \Leftrightarrow L(\boldsymbol{y} \mid \boldsymbol{x}) = e^{-\mathcal{L}(\boldsymbol{x})}.$$

With these, the acceptance ratio reduces to

$$A = \begin{cases} \min\left(e^{-\mathcal{L}(\boldsymbol{z})+\mathcal{L}(\boldsymbol{x}_{t-1})}, 1\right), \text{ if } \boldsymbol{z} \in S \\ 0, \text{ if } \boldsymbol{z} \notin S. \end{cases}$$

The loss function $\mathcal{L}$ is not limited to the standard loss function but rather can be any kind of norm-expression or sum of norm expression which characterizes the distance between the observed data and the model. This loss function can be used to promote different properties of the optimal solution, such as sparsity or smoothness, as explained in Section 3.1 when the loss function was first introduced.
The covariance $\boldsymbol{C}_t$ is defined as

$$\boldsymbol{C}_t = \begin{cases} \boldsymbol{C}_0, \ t \leq t_0 \\ s_d \text{cov}(\boldsymbol{X}_0, \ldots, \boldsymbol{X}_{t-1}) + s_d \epsilon \boldsymbol{I}_d, \ t > t_0, \end{cases}$$

where $\boldsymbol{C}_0$ is an initial covariance, t_0 is the length of an initial time period (burn-in and warmup), $\epsilon \geq 0$ is parameter that ensures that $\boldsymbol{C}_t$ is never singular, s_d is a scaling parameter given by $s_d = 2.4^2/d$ and $\boldsymbol{I}_d$ is a $d \times d$ unit matrix.

The initial covariance can be chosen freely but the closer it is to the converging value the shorter the burn-in period needs to be. The covariance $\text{cov}(\boldsymbol{X}_0, \ldots, \boldsymbol{X}_{t-1})$ can be estimated as the empirical (sample) covariance by using the empirical (sample) mean. The empirical mean $\overline{\boldsymbol{x}}_t \approx \mathbb{E}[\boldsymbol{X}_t]$ is computed by using previous values in the chain as

$$\overline{\boldsymbol{x}}_t = \frac{1}{t+1} \sum_{k=0}^{t} \boldsymbol{x}_k.$$

and the empirical covariance $\boldsymbol{Q}_t \approx \operatorname{cov}(\boldsymbol{X}_0, \ldots, \boldsymbol{X}_{t-1})$ can be computed as

$$\boldsymbol{Q}_t = \frac{1}{t-1}\sum_{k=0}^{t-1}(\boldsymbol{x}_k - \overline{\boldsymbol{x}}_{t-1})^{\mathrm{T}}(\boldsymbol{x}_k - \overline{\boldsymbol{x}}_{t-1}) = \frac{1}{t-1}\left(\sum_{k=0}^{t-1}\boldsymbol{x}_k^{\mathrm{T}}\boldsymbol{x}_k - t\overline{\boldsymbol{x}}_{t-1}^{\mathrm{T}}\overline{\boldsymbol{x}}_{t-1}\right).$$

As the chain becomes longer, estimating the covariance computationally heavier. Luckily, there exists a recursion relation both for $\overline{\boldsymbol{x}}_t$ and $\boldsymbol{C}_t$ which saves computation time.

Lemma 3.11. Consider an empirical mean

$$\overline{\boldsymbol{x}}_t = \frac{1}{t+1}\sum_{k=0}^{t}\boldsymbol{x}_k,$$

and an empirical covariance matrix

$$\boldsymbol{Q}_t = \frac{1}{t-1}\left(\sum_{k=0}^{t-1}\boldsymbol{x}_k\boldsymbol{x}_k^{\mathrm{T}} - t\overline{\boldsymbol{x}}_{t-1}\overline{\boldsymbol{x}}_{t-1}^{\mathrm{T}}\right).$$

The mean $\overline{\boldsymbol{x}}_t$ and the covariance $\boldsymbol{C}_t = s_d\boldsymbol{Q}_t + s_d\epsilon\boldsymbol{I}_d$ satisfy the recursion relations

$$\begin{aligned}
\overline{\boldsymbol{x}}_t &= \frac{1}{t+1}(t\overline{\boldsymbol{x}}_{t-1} + \boldsymbol{x}_t),\\
\boldsymbol{C}_{t+1} &= \frac{t-1}{t}\boldsymbol{C}_t + \frac{s_d}{t}\left(t\overline{\boldsymbol{x}}_{t-1}\overline{\boldsymbol{x}}_{t-1}^{\mathrm{T}} - (t+1)\overline{\boldsymbol{x}}_t\overline{\boldsymbol{x}}_t^{\mathrm{T}} + \boldsymbol{x}_t\boldsymbol{x}_t^{\mathrm{T}} + \epsilon\boldsymbol{I}_d\right).
\end{aligned}$$

Proof. The recursion for the empirical mean follows easily as

$$\overline{\boldsymbol{x}}_t = \frac{1}{t+1}\sum_{k=0}^{t}\boldsymbol{x}_k = \frac{1}{t+1}\left(\sum_{k=0}^{t-1}\boldsymbol{x}_k + \boldsymbol{x}_t\right) = \frac{1}{t+1}\left(t\overline{\boldsymbol{x}}_{t-1} + \boldsymbol{x}_t\right).$$

For the covariance, first note that

$$\sum_{k=0}^{t-1}\boldsymbol{x}_k\boldsymbol{x}_k^{\mathrm{T}} = \frac{t-1}{s_d}\boldsymbol{C}_t - (t-1)\epsilon\boldsymbol{I}_d + t\overline{\boldsymbol{x}}_{t-1}\overline{\boldsymbol{x}}_{t-1}^{\mathrm{T}}$$

from the definitions of $\boldsymbol{Q}_t$ and $\boldsymbol{C}_t$. By considering $\boldsymbol{C}_{t+1}$, it follows that

$$\begin{aligned}
\boldsymbol{C}_{t+1} &= s_d\boldsymbol{Q}_{t+1} + s_d\epsilon\boldsymbol{I}_d\\
&= \frac{s_d}{t}\left(\sum_{k=0}^{t}\boldsymbol{x}_k\boldsymbol{x}_k^{\mathrm{T}} - (t+1)\overline{\boldsymbol{x}}_t\overline{\boldsymbol{x}}_t^{\mathrm{T}}\right) + s_d\epsilon\boldsymbol{I}_d\\
&= \frac{s_d}{t}\left(\sum_{k=0}^{t-1}\boldsymbol{x}_k\boldsymbol{x}_k^{\mathrm{T}} + \boldsymbol{x}_t\boldsymbol{x}_t^{\mathrm{T}} - (t+1)\overline{\boldsymbol{x}}_t\overline{\boldsymbol{x}}_t^{\mathrm{T}}\right) + s_d\epsilon\boldsymbol{I}_d \quad \Big|\text{Substitute } \sum_{k=0}^{t-1}\boldsymbol{x}_k\boldsymbol{x}_k^{\mathrm{T}}\\
&= \frac{s_d}{t}\left(\frac{t-1}{s_d}\boldsymbol{C}_t - (t-1)\epsilon\boldsymbol{I}_d + t\overline{\boldsymbol{x}}_{t-1}\overline{\boldsymbol{x}}_{t-1}^{\mathrm{T}} + \boldsymbol{x}_t\boldsymbol{x}_t^{\mathrm{T}} - (t+1)\overline{\boldsymbol{x}}_t\overline{\boldsymbol{x}}_t^{\mathrm{T}}\right) + s_d\epsilon\boldsymbol{I}_d\\
&= \frac{t-1}{t}\boldsymbol{C}_t + \frac{s_d}{t}\left(t\overline{\boldsymbol{x}}_{t-1}\overline{\boldsymbol{x}}_{t-1}^{\mathrm{T}} + \boldsymbol{x}_t\boldsymbol{x}_t^{\mathrm{T}} - (t+1)\overline{\boldsymbol{x}}_t\overline{\boldsymbol{x}}_t^{\mathrm{T}} - (t-1)\epsilon\boldsymbol{I}_d + t\epsilon\boldsymbol{I}_d\right)\\
&= \frac{t-1}{t}\boldsymbol{C}_t + \frac{s_d}{t}\left(t\overline{\boldsymbol{x}}_{t-1}\overline{\boldsymbol{x}}_{t-1}^{\mathrm{T}} - (t+1)\overline{\boldsymbol{x}}_t\overline{\boldsymbol{x}}_t^{\mathrm{T}} + \boldsymbol{x}_t\boldsymbol{x}_t^{\mathrm{T}} - \epsilon\boldsymbol{I}_d\right)
\end{aligned}$$

which is the desired recursion relation for $\boldsymbol{C}_t$. □

If $\boldsymbol{C}_t$ does not have tendency to become singular, the AM-algorithm may be utilized with $\epsilon = 0$. More information about the algorithm is given in the paper by Haario et al. [11] in which they introduce the algorithm in detail.

After the AM-algorithm has been iterated long enough, the random samples in the chain will drawn from the stationary distribution of the chain. In Bayesian formalism, this stationary distribution is the posterior distribution i.e. the distribution of the parameters given the observed data. The samples drawn from the posterior distributions can be used to estimate the mean and the covariance matrix of the posterior.

In the case of the adaptive Metropolis algorithm, the mean estimation $\overline{\boldsymbol{x}}$ of the state vector is estimated as the empirical mean over the values in the chain after the burn-in period t_0. Similarly, the covariance matrix is estimated as the empirical covariance matrix over the values in the chain after the burn-in period t_0. The standard deviations for each component in $\overline{\boldsymbol{x}}$ is given by the square-root of the diagonal values of the empirical covariance matrix. Other possible error sources, which are not included during the fitting of the parameters, may be added included separately.

As for typical Markov chains, length of the burn-in period t_0 is arbitrary and depends mostly on the initial value and covariance matrix of the chain. A common strategy is first to obtain a good estimation for the state vector using another algorithm and then start the chain with that value. Similarly for the initial covariance matrix; starting with a good estimation for the covariance matrix allows shorter burn-in period.

Longer chains give more accurate estimations of the mean and the covariance matrix of the state vector but every iteration requires two forward model evaluations, which can make generating long chains slow depending on the evaluating time of the forward model. The strength of the AM-algorithm, or Markov chain Monte Carlo methods in general, is that the loss function has no requirements on differentiability. The estimation of the state vector obtained with the AM-algorithm is the posterior mean estimate which is different from the posterior mode estimate obtained by applying optimization algorithms, such as LM-algorithm, for the loss function. If the posterior is or is close to a multivariate normal distribution, the posterior mean and the posterior mode estimates are close to each other.

4. Data products

This chapter briefly introduces external data products used in this thesis. These products include data on the wind components as well as CO_2 satellite observations.

4.1 Orbiting Carbon Observatory-2

The Orbiting Carbon Observatory-2 (OCO-2) is a satellite launched on 2.7.2014 by NASA [3]. This thesis uses the OCO-2 version V10r lite files, which include bias correction and data screening and are available from NASA's Goddard Earth Sciences Data and Information Services Center. The OCO-2 satellite is capable of measuring CO_2 in a vertical atmospheric column with a high spatial resolution, usually eight measurements over its swath width, which is the width of the area on the Earth's surface seen by the satellite, if there are no clouds interfering with the measurement. For OCO-2, the swath width is about 10 kilometers. Methods, atmospheric sounding algorithms and other details used by the OCO-2 when performing retrievals are explained in detail in Algorithm Theoretical Basis Document by NASA's Jet Propulsion Laboratory [2]. Validation of the data collected by the OCO-2 are presented in the article by Wunch et al. [25] in which the OCO-2 data is compared with the data collected by ground based instruments.

If the OCO-2 flies by a strong enough emission source, an enhancement in the CO_2 field can be seen. A general overview of this type of measurement is illustrated in Figure 4.1.

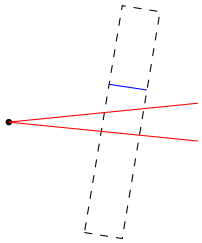


Figure 4.1: An overview of the OCO-2 observation. The dashed black rectangle marks the measured area, the black dot the plume source and the red lines represent the area where the enhancement of CO_2 can be observed. The blue line denotes the swath width of the satellite which is 10 km.

At each measurement location, the satellite records the dry air column average of CO_2 in parts per million (ppm) which is denoted as X_{CO_2}. The column average of any gas can be computed from the total column of gas with the formula [4]

$$X_{\text{gas}} = \frac{VC_{\text{gas}}/M_{\text{gas}}}{\frac{P_s}{g \cdot M_{\text{dry}}} - VC_{\text{H}_2\text{O}} \cdot \frac{M_{\text{H}_2\text{O}}}{M_{\text{dry}}}},$$

where VC_{gas} is the total vertical column of gas (kg/m^2), P_s is the surface pressure (Pa), g is the gravitational acceleration (m/s^2), $M_{\text{H}_2\text{O}}$, $M_{\text{H}_2\text{O}}$ and M_{gas} are molecular masses (g/mol) of dry air, water and gas of interest, respectively. The gravitational acceleration g is dependent on the latitude by the empirical formula

$$g(\phi) = 9.780327 \cdot (1 + 0.0053024 \cdot \sin^2(\phi) - 0.0000058 \cdot \sin(2\phi)),$$

where ϕ is the latitude for which of the location for which the column average is computed. As the OCO-2 measures the fraction of CO_2 to dry air, the total column of water vapor $VC_{\text{H}_2\text{O}}$ can be ignored and the column average can be computed as

$$X_{\text{gas}} = \frac{VC_{\text{gas}}}{\frac{P_s}{g \cdot M_{\text{dry}}/M_{\text{gas}}}} = \frac{g}{P_s} \cdot \frac{M_{\text{dry}}}{M_{\text{gas}}} \cdot VC_{\text{gas}}. \tag{4.1}$$

The column average of gas tells the fraction of that gas from the total vertical column of dry air. A column average of the CO_2 in the atmosphere is typically about 419 ppm as of writing this thesis but it slowly increases all the time due to the anthropogenic CO_2 emissions. The satellite measures the column average X_{CO_2}, but the models evaluate the total column VC_{CO_2} so the total column of CO_2 needs to be converted to ppm with the conversion factor in equation (4.1).

4.2 Atmospheric reanalysis 5

The fifth generation of atmospheric reanalysis (ERA5) is a comprehensive data set produced by the Copernicus Climate Change Service (C3S) at European Centre for Medium-Range Weather Forecasts (ECMWF) which includes many atmospheric variables. Some of these variables are given on different pressure levels [7] while the others are single level variables [8]. Variables available on different pressure levels include temperature, vorticity of air, divergence of air, wind components and many other similar types of variables. Single level variables include surface pressure, surface wind components, surface temperature, total column of water (vapor) and many others. The most useful regarding this thesis are wind components on surface and on pressure levels as well as surface pressures. Both the single level data and the pressure level data have on hourly temporal resolution and 0.01° longitudial and latitudial resolution. All the possible levels are between 100

hPa and 1 hPa (100 hPa being the standard surface pressure) with total of 37 layers and surface level data is typically a few metres above the Earth's surface.

Horizontal wind in longitudial and latitudial directions are given in units m/s but the vertical wind is given in Pa/s since the air is moving from pressure level to another. The wind in Pa/s can be converted to m/s by using Barometric formula and its inverse presented in equation (4.2) which describes the relation between the altitude h and the pressure p.

$$p(h) = p_0 \cdot \left(1 - \frac{Lh}{T_0}\right)^{\frac{gM_{\text{dry}}}{LR_0}} \Leftrightarrow h(p) = \frac{T_0}{L} \cdot \left(1 - \left(\frac{p}{p_0}\right)^{\frac{LR_0}{gM_{\text{dry}}}}\right) \tag{4.2}$$

where h is altitude, p is pressure, $p_0 = 101325$ Pa is the standard atmospheric pressure, $L \approx 0.00976$ K/m is temperature lapse rate (for dry air), $T_0 = 288.15$ K ($\approx 15°$C) is the standard temperature, $g = 9.8067$ m/s^2 is the gravitational acceleration, $M_{\text{dry}} = 0.0290$ kg/mol is molar mass of dry air and $R_0 = 8.3145$ is the universal gas constant. Using the chain rule and differentiating the function $h(p)$ with respect to time t yields

$$\frac{dh}{dt} = \frac{dh}{dp} \cdot \frac{dp}{dt} = -\frac{dp}{dt} \cdot \frac{T_0 R_0}{g M_{\text{dry}} p_0} \cdot \left(\frac{p}{p_0}\right)^{\frac{R_0 L}{gM_{\text{dry}}} - 1}$$

where the quantity dp/dt is the change in pressure with respect to the time which is given in the ERA5 data. It is worth noting that barometric formula is only valid inside troposphere which means that approximately the altitude must be less than 15 km and the pressure larger than 11.5 hPa. Since the plumes considered in this thesis stay inside the troposphere, using this kind of conversion is acceptable.

When using ERA5 data, an interpolation for the wind fields is usually required. Python's SciPy library has an `interpolate` module which is an efficient interpolation of points in $\mathbb{R}^n$ using known grid values. Alternatively, if a uncertainty propagation from the grid values to the interpolated value is relevant, a custom interpolation with Shepard's method described in Appendix B can also be used.

5. Results

This chapter introduces plume simulations, results and experiments with convection-diffusion model and Gaussian model. The considered plumes are originated from the Matimba power station which has latitude-longitude location $(-23.6686°, 27.6117°)$. Elevation is about 920 m above the sea level for the power plant and two chimneys (which are interpreted as a single source) are about 250 m high. When generating plumes with the convection-diffusion model, some initial velocity needs to be added for the plume, as explained in Appendix A. Adding the initial velocity allows the plume is able to spread further in shorter time. Initial velocities used on different dates are written in Table A.1.

5.1 Inversions with synthetic data

To test sensitivity of different model parameters, an observation can be simulated by generating a plume from a model. To avoid the inverse crime, the plume should be generated by using a different model than the model used for inversion. In this section, the simulated plume is created using the convection-diffusion model according to the wind fields from ERA5 data set and the inversions are performed with the Gaussian model. Generating a simulated plume allows a better control over the model parameters. Furthermore, the background noise will be added artificially so the mean and the standard deviation of the background can be controlled as well.

Figure 5.1 shows a simulated plume on 11.07.2018. After the plume has been generated, the background with the mean of 406.0 ppm and the standard deviation of 0.6 ppm is added to every pixel, representing the normal background noise of CO_2 in the atmosphere. The spatial resolution used for plume generation was originally $0.001°$ but each 10×10 pixel grid was averaged to form one larger pixel, making the spatial resolution $0.01°$. In order for the plume to spread further, some initial velocity has to be added, as explained in Appendix A. The added initial velocity and slowing scale of that initial velocity are determined according to Table A.1.

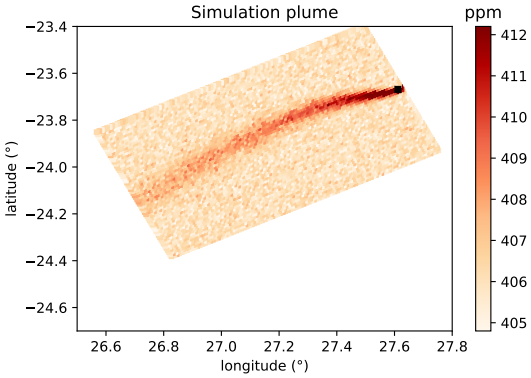
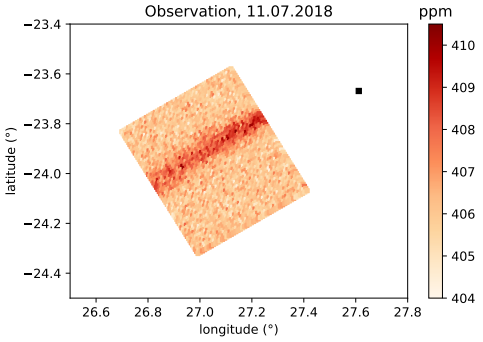


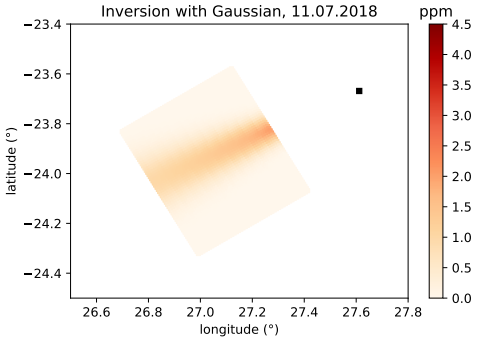
Figure 5.1: The simulated plume on 11.07.2018 using emission rate $5 \cdot 10^5$ g/s = 43.2 kt/d and the smallest allowed diffusion. The light orange rectangle presents the background values of the column average of CO_2, whereas the darker red values represent the simulated plume with higher values of the column average. The background values have a mean of 406 ppm and a standard deviation of 0.6 ppm.

The simulated plume can be sliced with an arbitrary observation rectangle to simulate an observation. The value representing a pixel in the artificial observation rectangle is chosen to be the closest known pixel from the generated plume pixels.

The first test is to fit the emission, wind direction and exponent using the Gaussian model with LM-algorithm. An artificial observation as well as the fitted plume after the inversion are presented in Figure 5.3. The width of this artificial observation is 0.5° which is much wider than the swath of the OCO-2 satellite. The observation rectangle is interpolated with 0.01° discretization and the model resolution is 0.005°.



(a) Simulated wide observation.



(b) Inversion using the Gaussian model.

Figure 5.2: Synthetic data inversion with the Gaussian model on 11.07.2018 with wide observation. The black square is the plume source and the lighter orange pixels present the background values of X_{CO_2}, whereas the darker red and orange values represent the simulated plume with higher values of X_{CO_2}. The background has a mean of 406.0 ppm and a standard deviation of 0.6 ppm.

The estimated emission rate is about 782000 g/s or 67.6 kt/d which is an over estimation compared to the true emission 43.2 kt/d. The wind direction is about $-2.70 \approx -154.7°$, the average wind speed u is 6.75 ± 0.32 m/s and the estimated stability parameter from the surface wind is estimated being about 207.5.

If the atmospheric stability parameter were to be fitted, it turns out that the stability parameter and the exponent have a strong negative correlation. This implies that they can not be reliably fitted together, even in the case of such a wide observation. A similar effect between these parameters is also present for observations from other dates. Since the stability parameter can be estimated from the surface wind, it is enough to estimate the stability parameter beforehand and then fit the exponent together with other the emission rate and the wind direction.

A simulated observation, closer to an observation measured by the OCO-2 satellite, is shown in Figure 5.3. This observation has width 0.17°, representing the realistic swath width of the OCO-2. As before, the spatial resolution of this simulated observation rectangle is 0.01° and the model resolution is 0.005°.

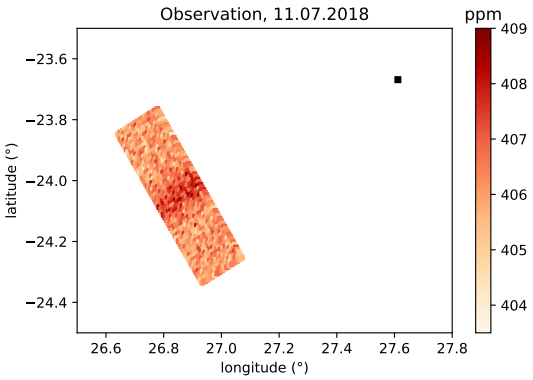


Figure 5.3: Simulated narrow observation. The black square is the plume source, and the orange rectangle presents the background values of the column average of CO_2, whereas the darker red values represent the simulated plume with higher values of the column average. The background values have a mean of 406.0 ppm and a standard deviation of 0.6 ppm.

The optimal state vector is now $\boldsymbol{x}_{\text{opt}} \approx (750000, -2.674, 0.866)$ which corresponds 64.8 kt/d for the emission rate, $-153.2°$ for the wind direction and 0.866 for the exponent. The estimated average wind speed 6.56 ± 0.31 m/s and the stability parameter is estimated

as $a \approx 207.6$. The covariance and the correlation matrices are

$$\boldsymbol{\Sigma}_{\boldsymbol{x}} \approx \begin{bmatrix} 1.10 \cdot 10^{9} & -6.59 \cdot 10^{-1} & 2.35 \cdot 10^{2} \\ -6.59 \cdot 10^{-1} & 2.06 \cdot 10^{-4} & -3.89 \cdot 10^{-8} \\ 2.35 \cdot 10^{2} & -3.89 \cdot 10^{-8} & 1.52 \cdot 10^{-4} \end{bmatrix}$$

$$\mathrm{corr}_{\boldsymbol{x}} \approx \begin{bmatrix} 1 & -0.001 & 0.578 \\ -0.001 & 1 & -0.002 \\ 0.578 & -0.002 & 1 \end{bmatrix}.$$

The wind direction has low correlation to both the emission and the exponent but the exponent clearly correlates with the emission rate. The estimated emission seems to overestimate the true emission rate 43.2 kt/d. The possible reasons for this can be uncovered by performing further test for the simulated data.

Since the information of the whole plume is available, the simulation rectangle can be moved freely further or closer to the source. This enables testing on how the distance affects the estimated emissions rate. The behavior of estimated mean values for the emission rate with respect to the distance between the observation and the source are shown in Figure 5.4.

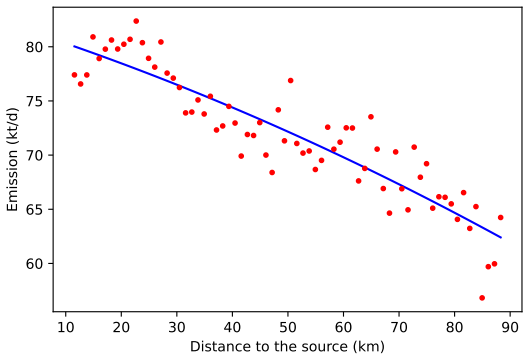


Figure 5.4: Emission rate estimation with respect to the distance between the observation and the source using Gaussian model and LM-algorithm. The red dots are the estimated parameter mean values and the blue line is a second degree polynomial fit.

As the distance gets smaller, the estimated emission seems to be growing. Optimally, the emission should remain about the same over all distances but clearly it is not true in this case. Possible reasons for this are that both or one of the use plume models are/is not consistent in distance. Most likely the problem is with the convection-diffusion model, as simulating the plume accurately is fairly complicated.

Other factors besides the fitted parameters affecting the inversion result are the background noise level and the resolution of the observation. For example, increase in the standard deviation of the background makes it harder to distinguish the plume from the

background. Naturally, this also makes optimizing the parameters more challenging and optimization algorithms may not be able to find reasonable model parameters. Figure 5.5 shows the behavior of the emission rate with respect to the background standard deviation.

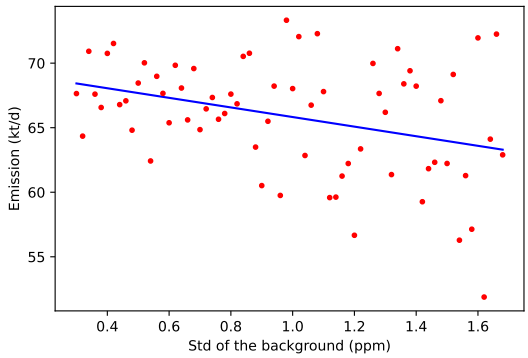


Figure 5.5: Emission rate estimation with respect to the background standard deviation using Gaussian model and LM-algorithm. The red dots are the estimated emission mean values and the blue line is a linear regression.

As can be seen, with small standard deviation of the background noise, the estimation is more consistent. When the noise level standard deviation rises, the emission estimation becomes increasingly more uncertain. This behavior is as expected as for larger background deviation, fitting a suitable exponents and finding a correct wind direction is harder, making also the emission estimation also more uncertain. This uncertainty could possibly be quantified and added as an error propagation for the estimated emission rate.

A similar effect for the retrieved parameters can be seen when the pixel size of modeled plume increases i.e. the model resolution decreases.

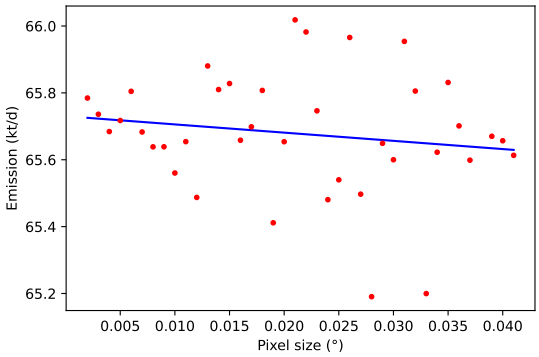


Figure 5.6: Emission rate estimation with respect to the background standard deviation using Gaussian model and LM-algorithm. The red dots are the estimated emission mean values and the blue line is a linear regression.

The behavior of the emission with respect to the pixel size is illustrated in Figure 5.6. As before, the inversion is more consistent with smaller pixels and when the pixel size increases, fitting the parameters becomes more unreliable due to lower amount of pixels in the modeled plume.

As can be seen from Figures 5.4, 5.5 and 5.6, the distance to the source, the standard deviation of the background noise level and the pixel size affect the quality of the obtained estimations for the emission rate. Experimenting on how these parameters affect the inversion is worth of studying but will not be considered further in this thesis.

5.2 Inversions with satellite data

Estimating emission rates by using measurements collected by the OCO-2 satellite will be considered next. These inversion can be done with both Gaussian and convection-diffusion model. Since the real data has no artificially added noise, the noise level needs to be estimated from the data.

A simple way of estimating the background noise can be done by using the pixels which are not part of the plume. Computing the mean and the standard deviation of the pixels far enough from the plume should give a reasonable estimation of the background, assuming that the other observed pixels do not have CO_2 enhancement caused by some other factor, like other plume source. A simple background noise estimation is illustrated in the Figure 5.7.

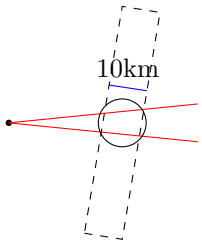


Figure 5.7: A simple background noise estimation. Pixels inside the red circle are considered as being a part of the plume and are ignored when computing the background estimation. The mean and the standard deviation of the Gaussian background are estimated using pixels outside the black circle.

Emission source considered in this thesis, the Matimba power station, is relatively remote location, far enough from other emission sources. Because of this, it is reasonable to assume that other pixels far enough from the plume are part of the normal background of X_{CO_2} in the atmosphere. Furthermore, this method also assumes that plume can be distinguished from the background; for larger sources such as Matimba, the enhancement is visually distinguishable. If the enhancement in X_{CO_2} is not significant enough, then

co-emitted gases, such as nitrogen dioxide (NO_2), can be used to determine the width and location of the plume. This related to the fact that emission rates for CO_2 and NO_2 are correlated as discussed by Hakkarainen et al. in their article [12].

With this background noise estimation, inversion for the OCO-2 observations can be performed. In order to compare the model and the observation visually, a cross-section of the plume is can be plotted. A cross-section is formed when observation and modelled pixels are projected to the center line of the satellite measurement. This is illustrated in the Figure 5.8.

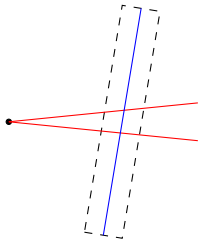


Figure 5.8: Cross section of the satellite measurement. Pixels in the observation are projected to the blue line, representing the cross section of the plume.

The observation to be considered next is a part of OCO-2 observation from date 04.05.2019. The observation is shown in Figure 5.9.

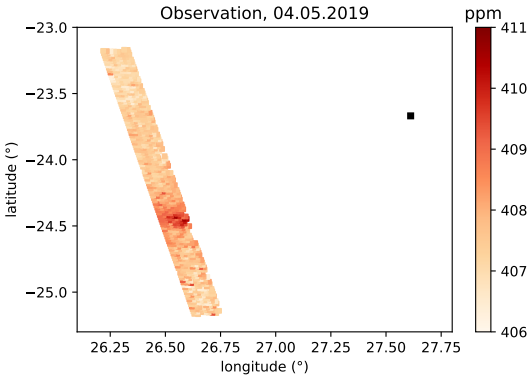


Figure 5.9: OCO-2 observation on 04.05.2019. The black square represents the location of the Matimba power station and the color map describes the observed enhancement of X_{CO_2} in ppm.

First, a cross-sectional fit using the Gaussian model with spatial resolution 0.005° with the optimal estimation is shown in Figure 5.10. The optimal estimation uses the default exponent $b = 0.894$ and wind direction $-152.0°$. The average wind speed is 4.31 ± 0.26 m/s, the stability parameter is estimated as 213 and the background noise 407.46 ± 0.49 ppm. These three quantities will be the same for each inversion.

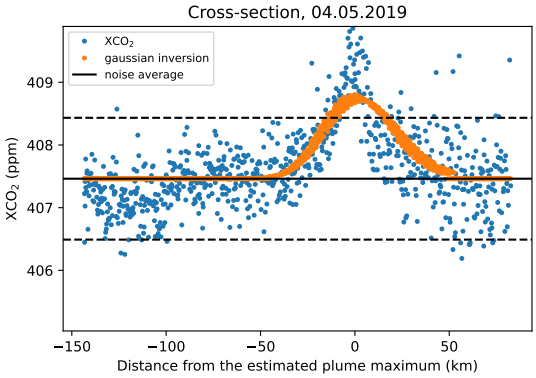


Figure 5.10: Linear optimal estimation with Gaussian model. Blue dots are the data points, orange dots is the modeled state, the black line is the noise average and black dashed lines are the 95% confidence interval for the Gaussian background noise.

The estimated emission rate is $x_{\text{opt}} \approx 1235000$ g/s and the standard deviation is 42400 g/s which correspond an emission rate of 106.7 ± 7.3 kt/d. The optimal estimation for linear may not be accurate as it only optimizes the emissions rate but is fast to perform, requiring only one model evaluation. The emission rate obtained from the linear optimal estimation can be used as the base line for further optimization with other algorithms.

The Levenberg–Marquardt algorithm can be used for optimizing the wind direction and the exponent, in addition to the emission rate. The cross-sectional fit using the Gaussian model and the Levenberg–Marquardt algorithm is shown in Figure 5.11.

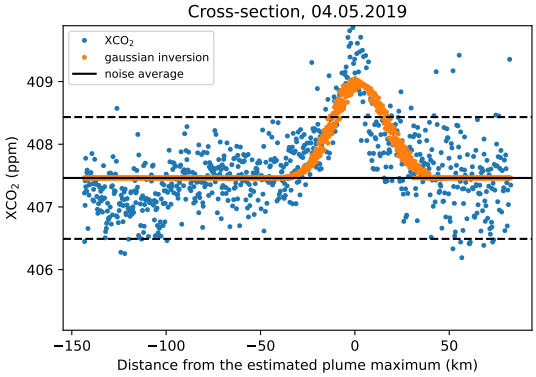


Figure 5.11: Levenberg–Marquardt optimization with Gaussian model. Blue dots are the data points, orange dots are the modeled state of the optimal state vector, the black line is the noise average and black dashed lines are the 95% confidence interval for the Gaussian background noise.

The optimal state vector is $\boldsymbol{x}_{\text{opt}} \approx (1066000, -2.505, 0.830)$ which corresponds 92.1 kt/d for the emission rate, $-143.5°$ for the wind direction and 0.830 for the exponent. The

covariance matrix of $\boldsymbol{x}_{\text{opt}}$ can be estimated using the Jacobian of the forward map J_F at the optimal state as $\boldsymbol{\Sigma}_{\boldsymbol{x}} \approx (J_F^T \boldsymbol{\Sigma}_{\text{obs}}^{-1} J)^{-1}$ as explained in Section 3.3. The covariance and correlation matrices are

$$\boldsymbol{\Sigma}_{\boldsymbol{x}} \approx \begin{bmatrix} 2.03 \cdot 10^9 & 2.11 \cdot 10^0 & 2.69 \cdot 10^2 \\ 2.11 \cdot 10^0 & 8.45 \cdot 10^{-6} & 6.98 \cdot 10^{-7} \\ 2.69 \cdot 10^2 & 6.98 \cdot 10^{-7} & 1.01 \cdot 10^{-4} \end{bmatrix},$$

$$\text{corr}_{\boldsymbol{x}} \approx \begin{bmatrix} 1 & 0.02 & 0.60 \\ 0.02 & 1 & 0.02 \\ 0.60 & 0.02 & 1 \end{bmatrix}.$$

Final method to estimate the parameters is the adaptive Metropolis algorithm. As explained in Section 3.5.2, the algorithm draws samples from the posterior distribution which can be used to estimate the posterior mean and the posterior covariance matrix. As this method is based on a random process, the results may vary a bit, depending on the length of the chain. The length of the chain used for this inversion is 5000, with 1000 as the burn-in period. Optimally, the chain generated by the random process should always converge to the same stationary distribution. A modeled state using the estimated posterior mean is shown in Figure 5.12.

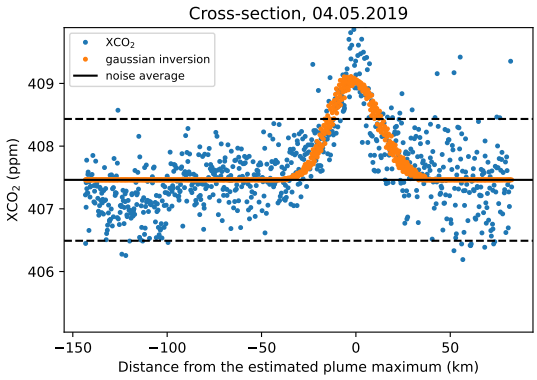


Figure 5.12: Sampling using the adaptive Metropolis algorithm with Gaussian model. Blue dots are the data points, orange dots is the modeled state of the posterior mean, the black line is the noise average and black dashed lines are the 95% confidence interval for the Gaussian background noise.

The state vector $\boldsymbol{x}_{\text{opt}} \approx (1047000, -2.526, 0.818)$ which corresponds 90.4 kt/d for the emission rate, $-144.7°$ for the wind direction and 0.818 for the exponent. An example of a random process or a random walk for each variable is shown in Figure 5.13.

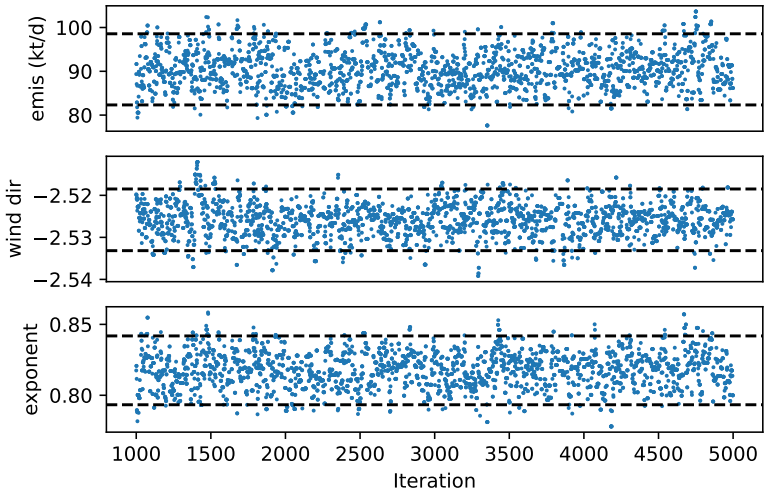


Figure 5.13: Random walk for each variable. After the burn-in, each variable varies randomly around the supposed mean with some standard deviation. The blue dots describe the values of the variable at each step after the initial 1000 long burn-in period. The dashed black lines describes the 95% confidence interval for each variable.

Furthermore, a covariance between each pair of variables can be visualized as in Figure 5.14. After the burn-in period, each variable is plotted as a function of every other variable, giving a visualization of the covariation between each pair of variables.

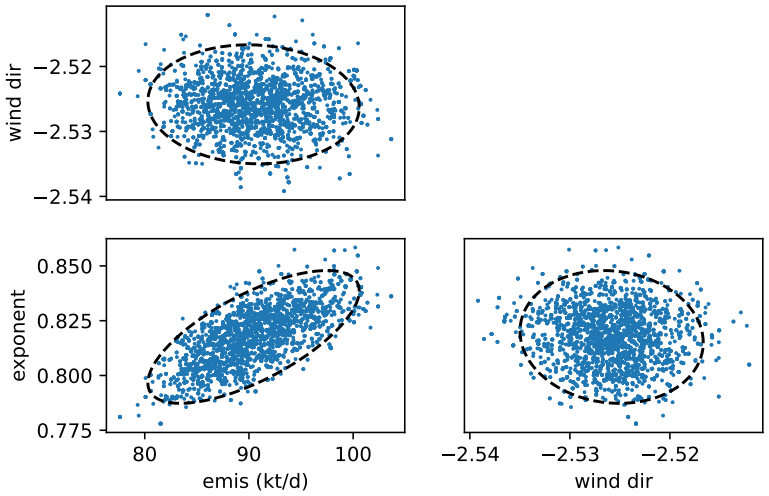


Figure 5.14: Pairwise covarying of variables. Blue dots describes the variables plotted against another variable and dashed black lines define the 95% confidence region for the bivariate normal distribution between each pair of variables.

For this specific run, the covariance and the correlation matrices are

$$\boldsymbol{\Sigma}_{\boldsymbol{x}} \approx \begin{bmatrix} 2.30 \cdot 10^{9} & -9.37 \cdot 10^{0} & 4.12 \cdot 10^{2} \\ -9.37 \cdot 10^{0} & 1.40 \cdot 10^{-5} & -3.96 \cdot 10^{-6} \\ 4.12 \cdot 10^{2} & -3.96 \cdot 10^{-6} & 1.53 \cdot 10^{-4} \end{bmatrix},$$

$$\mathrm{corr}_{\boldsymbol{x}} \approx \begin{bmatrix} 1 & -0.05 & 0.69 \\ -0.05 & 1 & -0.09 \\ 0.69 & -0.09 & 1 \end{bmatrix}.$$

The covariance and correlation matrices are fairly similar for between the LM-algoritm and AM-algorithm. The strong negative/positive correlations, as well as correlations close zero, coincide.

If the correlation between two variables approaches 1 and -1, then the confidence region ellipse squishes down closer to an increasing or a decreasing line, respectively, corresponding a linear regression line between the two variables. If the correlation approaches 0, then the ellipse approaches a circle, corresponding a bivariate Gaussian distribution of independent random variables.

Inversion using the convection-diffusion model for the observation 5.9 will be considered next. Evaluating the convection-diffusion model is more time taking compared to the Gaussian model. In fact, the convection-diffusion model with the chosen discretization of 0.005° takes far too long when using the adaptive Metropolis algorithm. Luckily, the Levenberg–Marquardt algorithm is manageable even with full temporal and spatial resolution, meaning 0.001° spatial and 1 second temporal discretization. Thus, only linear optimal estimation and Levenberg–Marquardt will be considered when inversions are performed with the convection-diffusion model.

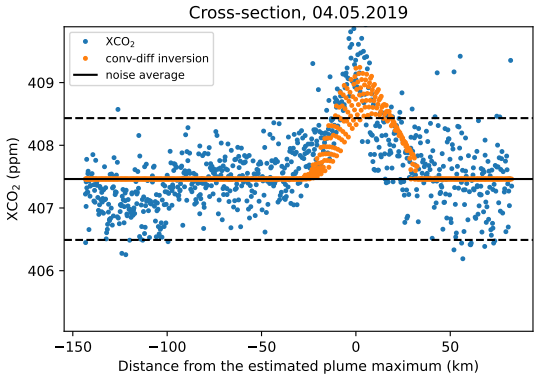


Figure 5.15: Linear optimal estimation with convection-diffusion model. Blue dots are the data points, orange dots are the modeled state of the optimal state vector, the black line is the noise average and black dashed lines are the 95% confidence interval for the Gaussian background noise.

Figure 5.15 shows a modeled state of the optimized emission rate using the optimal estimation for linear model. The estimated emission rate $x_{\text{opt}} = 526000$ g/s and the standard deviation 19000 g/s which for the emission rate corresponds 45.4 ± 3.2 (kt/d).

The magnitude of diffusion can be optimized using the Levenberg–Marquardt algorithm.

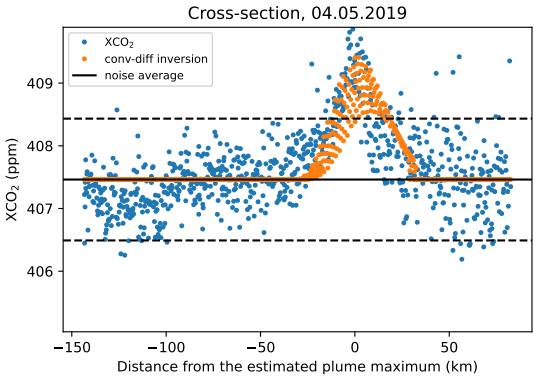


Figure 5.16: Levenberg–Marquardt optimization for linear model with convection-diffusion model. Blue dots are the data points, orange dots are the modeled state of the optimal state vector, the black line is the noise average and black dashed lines are the 95% confidence interval for the Gaussian background noise.

The modeled state using the state vector optimized with the LM-algorithm model is shown in Figure 5.16. The optimal vector $\boldsymbol{x}_{\text{opt}} \approx (524000, 0.875)$. The covariance and the correlation are

$$\boldsymbol{\Sigma}_{\boldsymbol{x}} \approx \begin{bmatrix} 9.03 \cdot 10^{8} & 2.28 \cdot 10^{3} \\ 2.28 \cdot 10^{3} & 8.98 \cdot 10^{-3} \end{bmatrix}$$

$$\text{corr}_{\boldsymbol{x}} \approx \begin{bmatrix} 1 & 0.80 \\ 0.80 & 1 \end{bmatrix}$$

which corresponds estimations 45.3 ± 3.7 kt/d for the emission and 0.875 ± 0.095 for the diffusion scale. For this date, the estimated emission rates with the linear optimal estimation and Levenberg–Marquardt are close to each other, meaning that the diffusion scale 1 is already a fairly good approximation. However, results clearly deviate from the emission rate about 90 kt/D obtained with the Gaussian model. This can happen due to many reasons but the most likely reason are the problems caused by the unexpected behavior of the plume simulation. This will be considered in more detail in Chapter 6.

The main interest is to estimate the emission rate using the OCO-2 data. Measuring the column average of CO_2 requires clear sky conditions and the observation should be close enough to the emission source. Hence, the OCO-2 data is viable to be used for the plume inversion only for certain dates. Moreover, the inversion may fail if variables have

too strong positive or negative correlation, if the background noise is too high or if there are too few pixels which include the plume [12].

Table 5.1 shows emission estimations done with the linear optimal estimation (OE), the Levenberg–Marquardt algorithm (LM) and adaptive Metropolis (AM). The value of retrieved emission rate as well as the root-mean-square error (RMSE) are reported. The root-mean-square error between the model and the data is computed as

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}{N}},$$

where y_i is an individual observation and $\hat{y}_i$ is the corresponding model value. The fitted model is better, the smaller the value of the RMSE is. Although different dates may have similar RMSE values, the fitted models may be completely different visually.

Date	Linear optimal estimation		Levenberg–Marquardt		Adaptive Metropolis	
	Emission	RMSE	Emission	RMSE	Emission	RMSE
02.06.2018	47.1 ± 5.0	0.519	80.1 ± 11.1	0.498	–	–
11.07.2018	81.9 ± 7.4	0.587	69.5 ± 7.3	0.582	76.5 ± 9.1	0.576
20.07.2018	66.6 ± 5.5	0.561	65.7 ± 7.0	0.561	67.1 ± 7.5	0.561
18.11.2018	50.2 ± 3.5	0.592	113.6 ± 6.6	0.470	110.6 ± 6.5	0.452
04.05.2019	106.7 ± 7.3	0.478	92.1 ± 6.7	0.471	90.4 ± 6.8	0.463
23.07.2019	34.8 ± 2.5	0.525	26.9 ± 2.3	0.518	27.2 ± 2.5	0.514
05.11.2019	60.9 ± 2.6	0.492	96.6 ± 4.6	0.423	98.3 ± 5.2	0.416
14.06.2020	32.4 ± 4.1	0.487	24.7 ± 3.38	0.479	29.0 ± 4.3	0.469
16.07.2020	95.0 ± 4.9	0.406	85.4 ± 5.1	0.400	88.6 ± 4.7	0.385
25.07.2020	38.6 ± 3.3	0.378	39.6 ± 3.8	0.378	39.0 ± 3.6	0.377
17.08.2020	113 ± 10.2	0.663	117.4 ± 12.1	0.663	139.8 ± 40.8	0.665
22.10.2020	21.0 ± 3.2	0.582	38.5 ± 5.9	0.542	40.5 ± 6.2	0.530

Table 5.1: Results using the Gaussian model. The emission rate is in kt/d and RMSE is in ppm. The error estimation is estimated from the covariance matrix and other error sources, such as uncertainty in the average wind speed, are added.

As can be seen, the RMSE is consistently the smallest with the adaptive Metropolis algorithm which indicates that it gives the best results. However, the RMSE of fitted models using Levenberg–Marquardt algorithm has only slight deviation from those of adaptive Metropolis so using only LM-algorithm is also justified due to it being faster than AM-sampling. The linear optimal estimation is good only if the optimal exponents happens to be close to the default exponent 0.894. The linear optimal estimation is very fast to run and offers an excellent way for visually estimating the optimal exponent when

comparing the cross-sectional fit with the observation. For date 02.06.2018, the AM-algorithm consistently fails to obtain a reasonable estimation for the posterior mean and for date 17.08.2020 the uncertainty can become very large when using the AM-algorithm.

Table 5.2 shows emission estimations done using convection-diffusion model. The optimal estimation fits the emission rate when diffusion scaling is exactly 1. The Levenberg–Marquardt algorithm fits the emission rate and the diffusion scale which is not always possible if the correlation is too strong. Since the convection-diffusion model is heavy to evaluate, the adaptive Metropolis algorithm was not tested as of writing these results. However, as stated earlier in the case of the Gaussian model, the Levenberg–Marquardt should also give reasonable results for the emission rates.

Date	Linear optimal estimation		Levenberg–Marquardt	
	Emission	RMSE	Emission	RMSE
02.06.2018	15.8 ± 1.8	0.535	–	–
11.07.2018	56.0 ± 5.1	0.589	56.1 ± 7.2	0.588
20.07.2018	65.7 ± 5.6	0.567	65.7 ± 11.3	0.567
18.11.2018	85.3 ± 4.4	0.462	85.3 ± 5.8	0.462
04.05.2019	45.4 ± 3.2	0.503	45.3 ± 3.7	0.500
23.07.2019	82.7 ± 5.9	0.519	81.4 ± 8.2	0.519
05.11.2019	87.7 ± 3.8	0.492	87.7 ± 4.8	0.492
14.06.2020	31.1 ± 5.9	0.505	–	–
16.07.2020	79.6 ± 4.3	0.437	79.6 ± 5.5	0.437
25.07.2020	84.2 ± 7.1	0.376	89.0 ± 10.3	0.376
17.08.2020	139.8 ± 14.8	0.705	–	–
22.10.2020	65.8 ± 9.8	0.551	71.0 ± 11.2	0.550

Table 5.2: Results using the convection-diffusion model. The emission rate is in kt/d and RMSE is in ppm. The error estimation is estimated from the covariance matrix and other error sources, such as the uncertainty in the wind field data, are added.

The RMSE is often the same between the two optimization methods as the default diffusion scale is usually a rather good fit. For some dates, the inversion with LM-algorithm fails due to strong positive or negative correlation between the emission rate and the diffusion scale, meaning that they can not be estimated together.

The estimated emission rates have reasonable magnitudes for each date but agree with the results retrieved from the Gaussian model only on some of the dates, such as 20.07.2018 and 05.11.2019. Possible reasons for some of the significant differences between Gaussian and convection-diffusion models are discussed in Chapter 6.

6. Discussion

Developing a mass transport models for gases and particles in the atmosphere is not an easy task as there is a lot of chemistry and physics involved. Everything happening in the atmosphere is impossible to model and some simplifications are always required. Even the implementation of the general convection-diffusion introduced in Section 2.1 is a very simple one and is mainly limited due to rather poor stability conditions and negative pixels appearing. Poor stability follows from the fact that the finite difference introduced in Section 2.1 is simple but not so reliable method for estimating derivatives when the function is discrete or has large jumps. Perhaps somehow smoothing or interpolating the concentration function would give better approximations for the derivatives, making the simulation more stable. Furthermore, the model requires adding the initial velocity and a slowing scale to be estimated which in general are challenging variables to both fit and justify. Finally, the reasons behind the inaccuracy between the total computed mass and the total expected mass are worth considering. The plume mass disappearing or appearing seemingly out of nowhere during the simulation may be caused by numerical errors, divergence of the wind field or there is something wrong in the implementation itself.

The convection-diffusion model ignores all the chemistry and relies on physical aspect of the mass transportation, namely wind fields and turbulence caused by the wind. The behavior of carbon dioxide (CO_2) has very little chemistry involved as CO_2 does not interact with other substances in the atmosphere. For other related gases such as nitrogen dioxide (NO_2), the model is not valid as NO_2 has a lifetime of few hours whereas CO_2 has lifetime of hundreds of years. If the model NO_2, or some other gas with a relatively short lifetime, is required to be simulated, then an exponential decaying term reducing the amount of the gas needs to be implemented to the simulation. Other physical aspects, which are ignored in the simulation, include the topography, curvature and rotation of Earth.

Naturally, more sophisticated mass transport models have already been developed. One of these models is called a FLEXible PARTicle dispersion model (FLEXPART) [18], originally released mid-1990s and written in Fortran language. This model is capable of simulating plumes accurately and takes into account stability conditions, chemical

reactions, wind fields and other similar variables.

Models which simulate plumes have potential to be very accurate, but also have tendency to become too heavy to evaluate in order for them to be used in optimization algorithms. Especially generating long random process chains with algorithms based on Monte Carlo simulation are impractical to be used with models which are heavy to evaluate. Due to this, many inversion methods do not actually generate the whole plume but focus on finding a good fit for the observed cross-section. Examples of these types of methods are the Gaussian plume model in Section 2.2, which fits a Gaussian function for the cross-section, or a cross-sectional flux method used in the article by Reuter et al. [20] which fits a combination of an exponential function and an affine function for the cross-section.

Different fitting algorithms or loss functions could also be considered. The optimal estimation for linear model in Section 3.3 and the Levenberg–Marquardt algorithm for non-linear model in Section 3.4 are based on the properties of l^2-norm and differentiability of the loss function. These gradient-based methods assumes that the (loss) function is differentiable which is a strong property for a function to have. A typical loss function given in the equation formulated in Equation (3.4) is defined true matrix products which also induces a (weighted) l^2-norm in $\mathbb{R}^n$ and is strongly related to the multivariate normal distribution 3.5. A function defined as the square of the l^2-norm is well-behaved and has simple expressions for the gradient and Hessian, allowing the use of the gradient-based methods. However, other kinds of loss functions could be used, like other l^p-norm expressions or adding different regularization terms to the standard l^2-loss function. Other kinds of loss function may not be differentiable so gradient-based algorithms are not valid. A concrete example of these kind of loss function is a function containing terms with l^1-norm as a function $\boldsymbol{x} \to ||\boldsymbol{x}||_1$ is not differentiable at the origin. As mentioned before, the adaptive Metropolis algorithm is not gradient based which could be used with various loss functions. Adding different kinds of regularization terms, prior information and trying out various norms could be worth of studying.

Finally, a method for finding plume trajectories explained in Appendix A provides a potentially useful way for simulating plumes. The current implementation of the convection-diffusion model is an Eulerian simulation method, meaning that the behavior of large number of particles in a fixed grid is traced. As mentioned before, a simple simulation using finite differences causes stability issues due to the limitations in the grid discretization. The alternative is to use Lagrangian simulation model, such as FLEXPART, in which single particles or a small batch of particles, are traced. Simulating single particles in flow and diffusion fields removes the spatial stability limitation as Lagrangian simulation does not require a fixed grid. However, concepts like non-linear diffusion equation and anomalous power law are involved in this kind of simulation method.

In simple terms, this means that the terms in the typical convection-diffusion equation (2.1) may be scaled with some exponential law, yielding to non-linear partial differential equations. More of these concepts are introduced in the article by Nazarenko and Grebenev [10]. In the end, implementing this type of simulation may be redundant since other models, such as FLEXPART, already exists. More useful task would be harnessing the FLEXPART model such that it could be efficiently used for performing similar inversion as has been done in this thesis.

Appendix A. Trajectory of plume

Trajectory is the path which a single particle, or a parcel, traces when moving in the flow field. Trajectories can be where the particle(s) of a plume end up after ejected to the atmosphere from the source when they move freely in the wind field. Studying trajectories is useful when figuring out how far and where the plume particles can reach after ejection.

The trajectory of a particle in a flow field $\boldsymbol{v}$ is can be described as a system of differential equations

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{v}(\boldsymbol{x}, t),$$

or written in components

$$\begin{cases} \frac{dx}{dt} = v_x(x, y, z, t), \\ \frac{dy}{dt} = v_y(x, y, z, t), \\ \frac{dz}{dt} = v_z(x, y, z, t). \end{cases}$$

This follows from the fact that the motion of the particle is determined purely by the flow field $\boldsymbol{v}$. Numerically solving this type of system of differential equations is straight forward. A simple iteration in order to track a single particle assuming the initial location (x_0, y_0, z_0) and time step τ proceeds as follows:

1. Interpolate an average wind vector $(\overline{v}_x, \overline{v}_y, \overline{v}_z)$ and uncertainties $\sigma_{v_x}, \sigma_{v_y}, \sigma_{v_z}$ (if available) from the wind field data at current location vector (x_i, y_i, z_i) of the particle.

2. Sample random all three wind components v_x, v_y, v_z from three normal distributions $\mathcal{N}(\overline{v}_x, \sigma^2_{v_x}), \mathcal{N}(\overline{v}_y, \sigma^2_{v_y}), \mathcal{N}(\overline{v}_z, \sigma^2_{v_z})$, correspondingly. If uncertainties are not available, use mean values for the wind components.

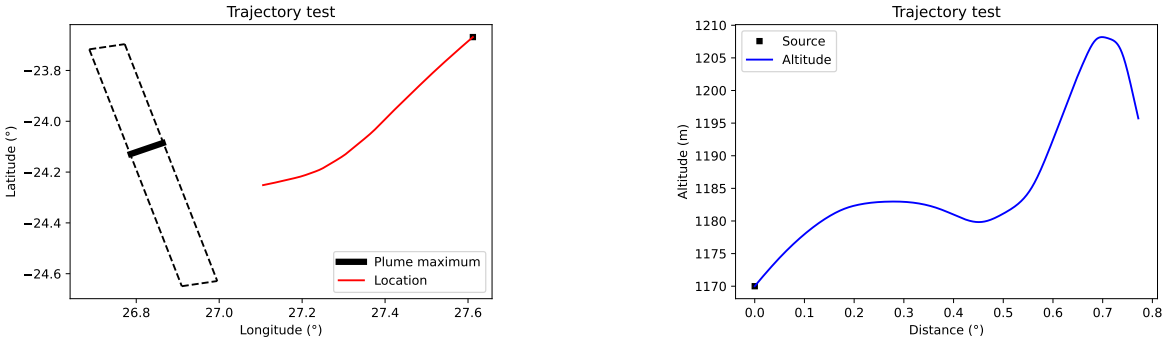
3. Update the location vector as

$$(x_{i+1}, y_{i+1}, z_{i+1}) = (x_i + \tau v_x, y_i + \tau v_y, z_i + \tau v_z).$$

4. Repeat until the desired simulation time is reached.

The uncertainty of the wind can be accounted with the deviations $\sigma_{v_x}, \sigma_{v_y}, \sigma_{v_z}$ but in this context, finding out the average trajectory of the particle is enough, meaning the the uncertainties may be ignored.

A test for a simple trajectory tracking is shown in Figure A.1. The wind used for this test are from date 11.07.2018 and the simulation time is from 05:04 to 12:04, where 12:04 corresponds the observation time for the OCO-2 measurement.



(a) Horizontal trajectory

(b) Altitude trajectory

Figure A.1: A simple trajectory tracking where the particle moves in longitude, latitude and altitude directions according to the wind field. On the left, the red line is the horizontal trajectory of the particle, the dashed rectangle is the OCO-2 observation and the black thick line is the estimated plume maximum. On the right, the blue line is the altitude trajectory of the particle. The black square is the source in both figures.

As can be seen, the horizontal trajectory aligns poorly with the estimated location where the plume should be seen. Similarly, the altitude seems to behave irregularly. A similar effect happens consistently for several dates when trajectories are tracked and for some dates, the particles may even hit the ground level. This leads to a conclusion that something is missing.

A correction to the trajectories can be done by adding some initial velocity for the particles. Adding some initial velocity is justified by the fact that the gas(es) emitted from chimneys is hot and tends to naturally rise upwards. When the emitted gas cools off, the effect of vertical wind gradually becomes the dominating factor in the vertical movement. The added initial vertical speed at the beginning of the simulation is assumed to decrease by an exponential law in which the the total vertical speed at time t is given by

$$v_z(t) = v_{z,\text{wind}}(t) + v_{z,\text{init}} \cdot e^{-s_t \cdot t},$$

where s_t is the slowing scale in time variable, $v_{z,\text{init}}$ is the initial vertical speed of the plume and $v_{z,\text{wind}}(t)$ is the vertical component of the wind field. As t grows, effect of the exponential terms gets smaller, eventually becoming insignificant.

The slowing effect can also be expressed with respect to the altitude z since the altitude is in direct relation to time t. With respect to the altitude, the slowing effect can be written as

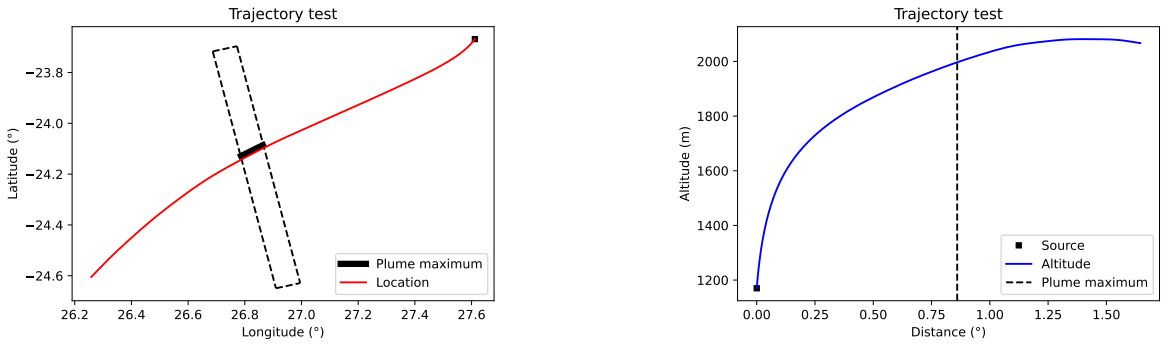
$$v_z(z) = v_{z,\text{wind}}(z) + v_{z,\text{init}} \cdot e^{-s_d \cdot d(z,z_s)}, \tag{A.1}$$

where s_d is the slowing scale in distance variable, $d(z, z_s)$ is the distance between z and the source z-component z_s and $v_{z,\text{wind}}(z)$ is the vertical wind at altitude z. Naturally,

the two terms in equation (A.1) have a strong correlation as both the initial velocity and the slowing scale affect the wind field component $v_{z,\text{wind}}(z)$. Conversely, the wind field component affects the distance $d(z, z_s)$ which affects the term $v_{z,\text{init}} \cdot e^{-s_d \cdot d(z,z_s)}$.

The initial speed and slowing scale are optimized such that the trajectory aligns as well as possible with the observed plume maximum. That is, the distance between the the trajectory line and the plume maximum is minimized. The optimization is done by using `minimize` function from SciPy's [24] optimization module which by default uses Nelder-Mean algorithm. The Nelder-Mead algorithm and its implementation is introduced in detail in the book "Numerical recipes" [19].

With added initial velocity $v_{z,\text{init}} \approx 0.5974$ m/s and slowing scale $s_d \approx 0.0044$ 1/m, a trajectory tracking using the wind fields on 11.07.2018 with simulation time from 05:04 to 12:04 is shown in Figure A.2.



(a) Horizontal trajectory

(b) Altitude trajectory

Figure A.2: Trajectory tracking with added initial velocity. On the left, the red line is the horizontal trajectory of the particle, the dashed rectangle is the OCO-2 observation and the black thick line is the estimated plume maximum. On the right, the blue line is the altitude trajectory of the particle and dashed black line is the location of the plume maximum. The black square is the source in both figures.

The horizontal trajectory now aligns much better with the estimated plume maximum. The reason for this is that the wind speed is usually faster with larger altitudes, allowing the plume more freely and faster. The altitude also behaves as expected; first the plume rises upwards due to air being hot, but then starts to cool off, causing the plume rising starting to decay. After a long enough time, the behavior of the particle is entirely determined by the wind field.

Different initial velocities and slowing scales used on different dates are listed in Table A.1.

Date	$v_{z,\text{init}}$ (m/s)	s_d (1/m)
02.06.2018	0.6794	0.0039
11.07.2018	0.5974	0.0044
20.07.2018	0.9999	0.0070
18.11.2018	1.0004	0.0068
04.05.2019	0.7500	0.0050
23.07.2019	0.9993	0.0250
05.11.2019	1.0000	0.0050
14.06.2020	1.1500	0.0190
16.07.2020	0.9975	0.0071
25.07.2020	1.0018	0.0047
17.08.2020	0.2200	0.0350
22.10.2020	1.4100	0.0010

Table A.1: Values used for initial velocities and slowing scales on different dates.

For some dates, the observation is fairly close to the source in which case the initial velocity and the slowing scale affect the plume relatively little. If the observation is far from the source and the simulation time is relatively long, then optimizing these variables becomes important so that the trajectory aligns well with the observed plume.

With the help of trajectories, the simulation can be reduced from 3D to 2D by averaging wind fields on different layers. First, an interpolating function for the altitude with respect to horizontal distance is determined using `interp1d` function from SciPy's [24] interpolation module. At each longitude-latitude location, the horizontal distance to the source is computed, then the corresponding altitude is interpolated with the interpolating function. The average wind field value at the longitude-latitude location is given by interpolating gridded wind field data to the (longitude,latitude,altitude) coordinate with a function `interpnd` from SciPy's [24] interpolation module. This method of averaging accounts the altitude aspect of the plume without actually simulating the plume in 3D, saving memory and computation time.

The average wind speed u in the Gaussian model (2.2) can also be estimated by using the averaged wind fields over altitude layers. First, wind fields at the nearest known time to the starting time are retrieved from an external data set. Then the wind fields are averaged over the altitude layers using the interpolating function which tells the altitude at given distance. The average wind speed of the plume is computed from the averaged wind field by using the pixels near the plume center line. Averaging these pixels together gives an estimation for an average wind speed at the starting time. Repeating this over each known time given in the data, an estimation for the average wind speed at each

time in the plume region is obtained. Averaging these together gives an estimation for the average wind speed in the plume region over the entire simulation time.

Appendix B. Shepard's interpolation

Shepard's method or Shepard's interpolation is a simple interpolation based on inverse distance weighting. Given a set of points on a metric space, a new value can be interpolated to any location using the known points such that the weights are determined by the distances to each of those points. If the distance increases, then the corresponding weight approaches zero and if the distance get smaller, the corresponding weight approaches one. The Shepard's interpolation can be formulated as in Theorem B.1.

Theorem B.1. *Consider points* $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^n$ *and* $\boldsymbol{x} \in \mathbb{R}^n$. *Given function values* $f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_N)$, *an interpolated value* $f(\boldsymbol{x})$ *is given by*

$$f(\boldsymbol{x}) = \begin{cases} \frac{\sum_{i=1}^N w_i(\boldsymbol{x}) f(\boldsymbol{x}_i)}{\sum_{i=1}^N w_i(\boldsymbol{x})}, & \text{if } d(\boldsymbol{x}, \boldsymbol{x}_i) \neq 0 \text{ for all } i \\ f(\boldsymbol{x}_i) \text{ if } d(\boldsymbol{x}, \boldsymbol{x}_i) = 0 \text{ for some } i \end{cases}, w_i(\boldsymbol{x}) = \frac{1}{d(\boldsymbol{x}, \boldsymbol{x_i})^p}$$

where d *is the metric function and* $p \geq 1$ *is the smoothing parameter.*

Proof. Define a functional

$$\mathcal{F}(\boldsymbol{x}, y) = \begin{cases} \left(\sum_{i=1}^N \frac{(y - f(\boldsymbol{x}_i))^2}{d(\boldsymbol{x}, \boldsymbol{x}_i)^p} \right)^{1/p}, & \text{if } \boldsymbol{x} \neq \boldsymbol{x}_i \text{ for all } i \\ 0, & \text{if } \boldsymbol{x} = \boldsymbol{x}_i \text{ for some } i \end{cases}$$

which describes the total deviation between the interpolating function $y = y(\boldsymbol{x})$ and the interpolated points $f(\boldsymbol{x}_i)$ when weighted with inverse distances. If $\boldsymbol{x} = \boldsymbol{x}_i$ for some i, then the deviation is zero and the minimizing value is trivially $y = f(\boldsymbol{x}_i)$. If $\boldsymbol{x} \neq \boldsymbol{x}_i$ for all i, then differentiating $\mathcal{F}$ with respect to y and setting it to zero yields

$$\begin{aligned} &\frac{\partial \mathcal{F}}{\partial y} = \sum_{i=1}^N 2 \cdot \frac{y - f(\boldsymbol{x}_i)}{d(\boldsymbol{x}, \boldsymbol{x}_i)^p} \cdot \frac{1}{p} \left(\sum_{i=1}^N \frac{(y - f(\boldsymbol{x}_i))^2}{d(\boldsymbol{x}, \boldsymbol{x}_i)^p} \right)^{1/p - 1} = 0 \\ &\Leftrightarrow \sum_{i=1}^N \frac{y - f(\boldsymbol{x}_i)}{d(\boldsymbol{x}, \boldsymbol{x}_i)^p} = 0 \\ &\Leftrightarrow \sum_{i=1}^N \frac{y}{d(\boldsymbol{x}, \boldsymbol{x}_i)^p} = \sum_{i=1}^N \frac{f(\boldsymbol{x}_i)}{d(\boldsymbol{x}, \boldsymbol{x}_i)^p} \\ &\Leftrightarrow y \sum_{i=1}^N w_i(\boldsymbol{x}) = \sum_{i=1}^N f(\boldsymbol{x}_i) w_i(\boldsymbol{x}) \\ &\Leftrightarrow y = \frac{\sum_{i=1}^N f(\boldsymbol{x}_i) w_i(\boldsymbol{x})}{\sum_{i=1}^N w_i(\boldsymbol{x})}, \end{aligned}$$

which is the desired function $y = y(\boldsymbol{x}) = f(\boldsymbol{x})$ to minimize the deviation. □

Since weights in the Shepard's method include the inverse of the distance, a floating point overflow may occur when diving with a distance close to a zero. To avoid this, the numerator and the denominator of the expression

$$\frac{\sum_{i=1}^{N} w_i(\boldsymbol{x}) f(\boldsymbol{x}_i)}{\sum_{i=1}^{N} w_i(\boldsymbol{x})}$$

can be multiplied with a product over all pth powers of distances between $\boldsymbol{x}$ and $\boldsymbol{x}_i$. That is,

$$\frac{\sum_{i=1}^{N} w_i(\boldsymbol{x}) f(\boldsymbol{x}_i)}{\sum_{i=1}^{N} w_i(\boldsymbol{x})} = \frac{\prod_{j=1}^{N} d(\boldsymbol{x}, \boldsymbol{x}_j)^p \sum_{i=1}^{N} w_i(\boldsymbol{x}) f(\boldsymbol{x}_i)}{\prod_{j=1}^{N} d(\boldsymbol{x}, \boldsymbol{x}_j)^p \sum_{i=1}^{N} w_i(\boldsymbol{x})} = \frac{\sum_{i=1}^{N} \prod_{j=1, j\neq i}^{N} d(\boldsymbol{x}, \boldsymbol{x}_j)^p f(\boldsymbol{x}_i)}{\sum_{i=1}^{N} \prod_{j=1, j\neq i}^{N} d(\boldsymbol{x}, \boldsymbol{x}_j)^p}.$$

In this form, the weight for the value $f(\boldsymbol{x}_i)$ is given by

$$w_i = \frac{\prod_{j=1, j\neq i}^{N} d(\boldsymbol{x}, \boldsymbol{x}_j)^p}{\sum_{i=1}^{N} \prod_{j=1, j\neq i}^{N} d(\boldsymbol{x}, \boldsymbol{x}_j)^p}, \tag{B.1}$$

and trivially

$$\sum_{i=1}^{N} w_i = 1.$$

A metric d is naturally the Euclidean distance. The smoothing parameter controls the smoothness of the interpolation. As an example, consider an interpolation in $\mathbb{R}$ with control points x_1, x_2 and x_3 with corresponding values y_1, y_2 and y_3.

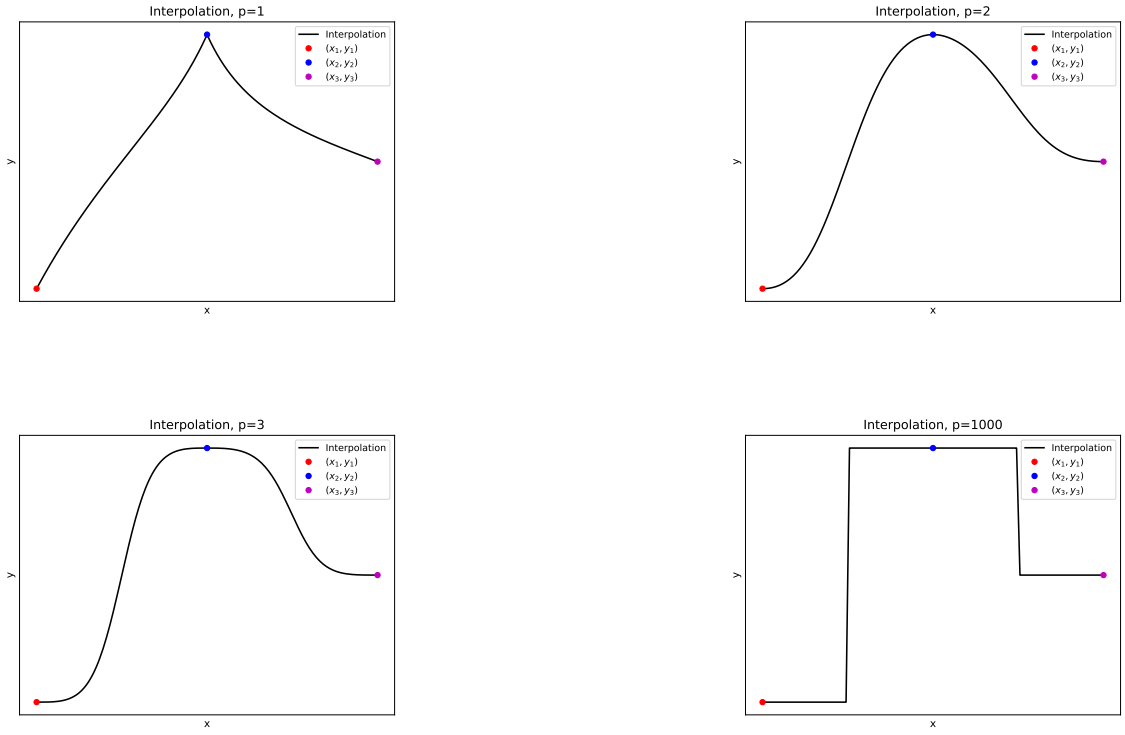


Figure B.1: Shepard's interpolations in 1D with different values of p. The points with matching colours stay at the same corresponding places, while the curve changes.

Interpolations with different values of p are shown in Figure B.1. The interpolation with $p = 1$ is close to linear interpolation. In fact, if there were only two points, the interpolation would be the linear interpolation (line) between the points. With $p = 2$, the interpolation is smooth which a very useful property to have in general. With $p = 3$ the interpolation is also smooth but starts to get steeper from one point to another. As p increases, the interpolates approaches the nearest point interpolation, corresponding interpolation in l^∞-norm, which means that the interpolated value has the same value as the nearest point. By default, the interpolation using the Shepard's method will be done with $p = 2$. This is a natural choice since $p = 2$ is strongly related to Euclidean distance.

As an another example of the Shepard's interpolation, consider a square in $\mathbb{R}^2$. The corners of the square are $(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)$ and their corresponding values are z_1, z_2, z_3, z_4 with uncertainties (standard deviations) $\sigma_1, \sigma_2, \sigma_3, \sigma_4$. The goal is to interpolate a value to a coordinate (x, y) inside the square using the four corner points and find a reasonable uncertainty estimation for the interpolated value. Distances to each corner are given by the standard Euclidean distance $d_i = \sqrt{(x - x_i)^2 + (y - y_i)^2}$. The situation is illustrated in Figure B.2.

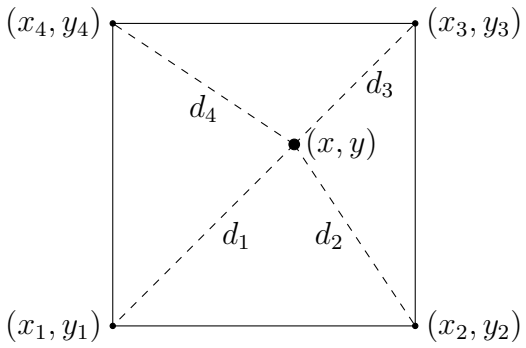


Figure B.2: Shepard's interpolation in a square using corner values.

By utilizing the Shepard's interpolation, according to the formula B.1 the weights with $p = 2$ are

$$
\begin{aligned}
w_1 &= \frac{(d_2 d_3 d_4)^2}{(d_2 d_3 d_4)^2 + (d_1 d_3 d_4)^2 + (d_1 d_3 d_4)^2 + (d_1 d_2 d_3)^2} \\
w_2 &= \frac{(d_1 d_3 d_4)^2}{(d_2 d_3 d_4)^2 + (d_1 d_3 d_4)^2 + (d_1 d_3 d_4)^2 + (d_1 d_2 d_3)^2} \\
w_3 &= \frac{(d_1 d_2 d_4)^2}{(d_2 d_3 d_4)^2 + (d_1 d_3 d_4)^2 + (d_1 d_3 d_4)^2 + (d_1 d_2 d_3)^2} \\
w_4 &= \frac{(d_1 d_2 d_3)^2}{(d_2 d_3 d_4)^2 + (d_1 d_3 d_4)^2 + (d_1 d_3 d_4)^2 + (d_1 d_2 d_3)^2}.
\end{aligned}
$$

and thus the interpolated value z at (x, y) is

$$z = w_1 z_1 + w_2 z_2 + w_3 z_3 + w_4 z_4.$$

It is clear that if $d_i = 0$ for some i, then $w_i = 1$, $w_j = 0$ for $j \neq i$ and the interpolated value $z = z_i$. Furthermore, if $d_1 = d_2 = d_3 = d_4$, then clearly $w_1 = w_2 = w_3 = w_4 = 1/4$ and the interpolation reduces to the standard arithmetic mean. The interpolation in Figure B.2 is not continuous over the boundary of the square as the points used for computing the weights change when moving over the boundary. An easy fix is to use larger number of surrounding points for the interpolation. If all available grid points are used for computing the weights for the interpolated value, then the interpolating function will be at least continuous inside the region defined by the grids. The smoothness of derivatives of this interpolating function will not be considered further.

With an assumption that the values in the corners have no covariance, the variance of z is given by

$$\sigma_z^2 = w_1^2\sigma_1^2 + w_2^2\sigma_2^2 + w_3^2\sigma_3^2 + w_4^2\sigma_4^2.$$

If pair-wise grid values have covariance, then the variance of the z is given by

$$\begin{aligned}\sigma_z^2 = {} & w_1^2\sigma_1^2 + w_2^2\sigma_2^2 + w_3^2\sigma_3^2 + w_4^2\sigma_4^2 \\ & + 2w_2w_1\mathrm{cov}(z_2, z_1) + 2w_3w_1\mathrm{cov}(z_3, z_1) + 2w_3w_2\mathrm{cov}(z_3, z_2) \\ & + 2w_4w_1\mathrm{cov}(z_4, z_1) + 2w_4w_2\mathrm{cov}(z_4, z_2) + 2w_4w_3\mathrm{cov}(z_4, z_3).\end{aligned}$$

In a general case, if the set of points used for interpolation have known covariances and variances, then the variance of the interpolated value is

$$\begin{aligned}\sigma_z^2 &= \mathrm{var}\left(\sum_{i=1}^{N} w_i z_i\right) \\ &= \sum_{i=1}^{N} \mathrm{var}\left(w_i z_i\right) + 2\sum_{i=2}^{N}\sum_{j<i} \mathrm{cov}\left(w_i z_i, w_j z_j\right) \\ &= w_i^2 \sum_{i=1}^{N} \mathrm{var}\left(z_i\right) + 2\sum_{i=2}^{N}\sum_{j<i} w_i w_j \mathrm{cov}(z_i, z_j).\end{aligned}$$

Shepard's interpolation is useful whenever the uncertainty of the interpolated value is relevant.

Appendix C. Plume inversion program

This appendix briefly explains the architecture of the Python program used for plume inversions. The program is written in such a way that adding extensions, such as handling more plume sources, and modifying individual functions should be easy to do. The plume inversion program and other realted scripts used in this thesis can be found on GitLab https://gitlab.com/Jeann96/plume_inversion.

The program relies mostly on Python's NumPy-library [13] which enables vectorized list operations using NumPy-array objects; handling arrays with large amounts of elements is much faster with numpy than with the basic Python. The other important library is `scipy` [24] which offers various interpolation, optimization and other useful tools related to scientific computing.

The program consists of different modules (files), each containing various functions used during the plume inversion. The `plume_inversion_functions.py` contains the most relevant functions for the plume inversion such as the primary inversion function `plume_inversion`, optimization algorithms like Adaptive Metropolis, background noise estimation, wind direction and speed estimation, single particle simulations and other similar functions. Module `data_handling.py` is similar to the module `plume_inversion_functions.py`, but contains more generic data handling functions such as pixel averaging, projection, distance computations, custom interpolations etc. as well as functions to read observation data from a local file. The module `CDS_handling.py` handles the reading and interpolation of Copernicus data sets (ERA5 data). If the local data file is missing, then a server request is executed and data will downloaded from the server. All data input files will be stored in the folder `input_data`.

The model files `gaussian_model.py` and `convection_diffusion_model.py` contain functions which evaluate the corresponding plume models given the input parameters. These functions are called whenever a model evaluation is required. Optimizing these functions is very important as they are usually called many times during the inversion process. Adding a new model as a Python function and using it for the inversion is straight forward.

The rest of the modules are shorter, containing only a few functions. The module `atmospheric_functions.py` contains atmposheric formulas mostly for converting a given pressure to altitude and vice versa. The module `location_params.py` only contains a dictionary which includes information on the location for which the plume inversion is

performed. The information includes the coordinates for the sources and altitude of the surface from the sea level near the source. Currently, the Matimba power station is the only location for which the inversions are performed and tested. The final module which is a part of the plume inversion is `plume_plotting.py`. This module contains tools for visualizing observations and fitted plumes. The plotting tools use Python's Matplotlib-library for plotting which is the default plotting tool used with Python language.

The main file of the plume inversion program is `plume_inversion_main.py` in which parameters are defined by the user, input data sets are read, plume inversion is executed. The main functions prints, plots and saves all the results into the `results` folder. A simple flow chart of the main program is shown in Figure C.1.

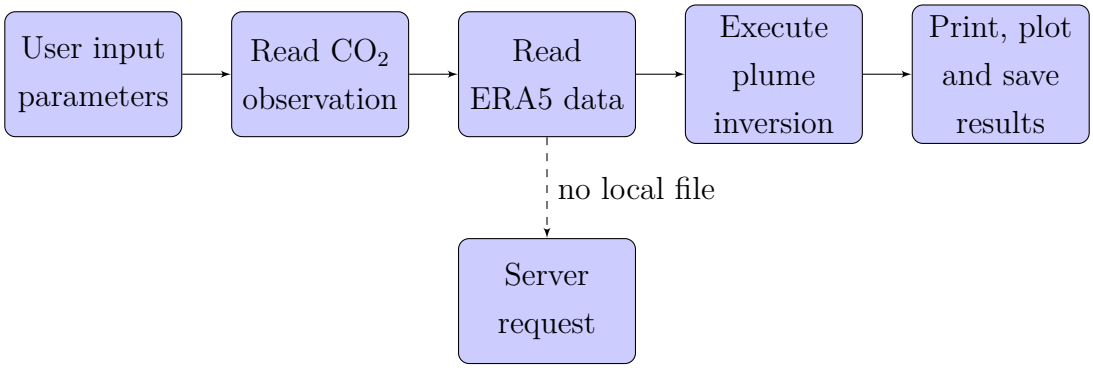


Figure C.1: A flow chart representation of the plume inversion program.

The most tasking part is performing the plume inversion, as it involves various details and algorithms. A flow chart of the plume inversion program is shown in Figure C.2.

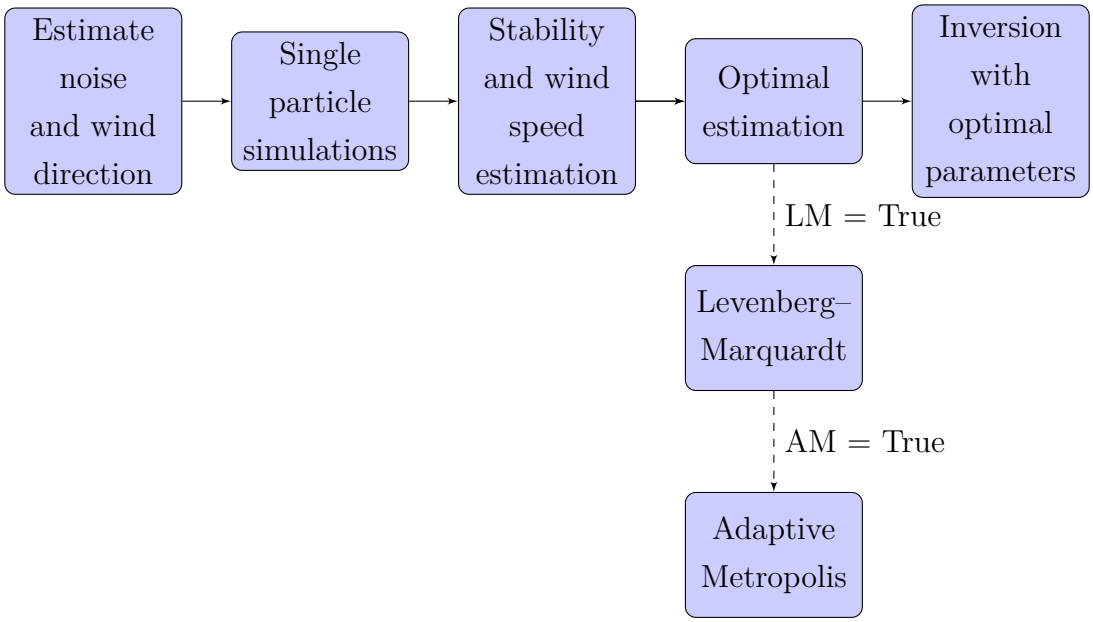


Figure C.2: A flow chart representation of the plume inversion function.

Further details of the code are left out as they are out of the scope of this thesis. Furthermore, the code will keep updating and the smallest details will most certainly change.

In addition to the plume inversion program, there are also modules related to constructing simulated plumes. The main program `simulation_main.py` can be used for creating a simulated plume using one of the implemented plume models. Alternative, plumes can also be constructed from an existing plume data by interpolating the data to desired coordinates using functions from `simulation_functions.py`. The final additional file is `testing.py` which can be used to perform various test for the plume inversion with different input parameters, as has been done in Section 5.1.

References

[1] Air Resources Laboratory. Pasquill stability classes. https://www.ready.noaa.gov/READYpgclass.php. Accessed May 5, 2022.

[2] H. Boesch, L. Brown, R. Castano, M. Christi, B. Connor, D. Crisp, A. Eldering, B. Fisher, C. Frankenberg, M. Gunson, et al. Orbiting Carbon Observatory–2 Level 2 Full Physics Algorithm Theoretical Basis Document, JPL document OCO D-55207. *Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA*, 2015.

[3] D. Crisp, H. R. Pollock, R. Rosenberg, L. Chapsky, R. A. Lee, F. A. Oyafuso, C. Frankenberg, C. W. O'Dell, C. J. Bruegge, G. B. Doran, et al. The on-orbit performance of the Orbiting Carbon Observatory-2 (OCO-2) instrument and its radiometrically calibrated products. *Atmospheric Measurement Techniques*, 10(1):59–81, 2017.

[4] N. M. Deutscher, D. W. T. Griffith, G. W. Bryant, P. O. Wennberg, G. C. Toon, R. A. Washenfelder, G. Keppel-Aleks, D. Wunch, Y. Yavin, N. T. Allen, J.-F. Blavier, R. Jiménez, B. C. Daube, A. V. Bright, D. M. Matross, S. C. Wofsy, and S. Park. Total column CO_2 measurements at Darwin, Australia — site description and calibration against in situ aircraft profiles. *Atmospheric Measurement Techniques*, 3(4):947–958, 2010.

[5] R. M. Dudley. *Real analysis and probability*. CRC Press, 2018.

[6] R. Durrett. *Probability: Theory and Examples*, volume 49 of *Cambridge series in statistical and probabilistic mathematics*. Cambridge University Press, Cambridge, 2019.

[7] ECMWF. ERA5 hourly data on pressure levels. https://www.ecmwf.int/en/forecasts/dataset/ecmwf-reanalysis-v5. Accessed May 5, 2022.

[8] ECMWF. ERA5 hourly data on single levels. https://cds.climate.copernicus.eu/cdsapp#!/dataset/reanalysis-era5-single-levels?tab=form. Accessed May 5, 2022.

[9] L. C. Evans. *Partial differential equations*. Graduate studies in mathematics ; volume 19. American Mathematical Society, Providence, R.I, 2nd edition, 2010.

[10] V. Grebenev, S. Nazarenko, S. Medvedev, I. Schwab, and Y. A. Chirkunov. Self-similar solution in the Leith model of turbulence: anomalous power law and asymptotic analysis. *Journal of Physics A: Mathematical and Theoretical*, 47(2):025501, 2013.

[11] H. Haario, E. Saksman, and J. Tamminen. An adaptive Metropolis algorithm. *Bernoulli*, 7(2):223–242, 04 2001.

[12] J. Hakkarainen, M. E. Szeląg, I. Ialongo, C. Retscher, T. Oda, and D. Crisp. Analyzing nitrogen oxides to carbon dioxide emission ratios from space: A case study of Matimba Power Station in South Africa. *Atmospheric Environment: X*, 10:100110, 2021.

[13] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020.

[14] J. L. Mueller and S. Siltanen. *Linear and nonlinear inverse problems with practical applications*. Society for Industrial and Applied Mathematics (SIAM), 2012.

[15] R. Nassar, T. G. Hill, C. A. McLinden, D. Wunch, D. B. A. Jones, and D. Crisp. Quantifying CO_2 Emissions From Individual Power Plants From Space. *Geophysical Research Letters*, 44(19):10,045–10,053, 2017.

[16] R. Nassar, J.-P. Mastrogiacomo, W. Bateman-Hemphill, C. McCracken, C. G. MacDonald, T. Hill, C. W. O'Dell, M. Kiel, and D. Crisp. Advances in quantifying power plant CO_2 emissions with OCO-2. *Remote Sensing of Environment*, 264:112579, 2021.

[17] R. C. O'Reilly and J. M. Beck. A family of large-stencil discrete Laplacian approximations in three-dimensions. *Int. J. Numerical Methods Eng*, pages 1–16, 2006.

[18] I. Pisso, E. Sollum, H. Grythe, N. I. Kristiansen, M. Cassiani, S. Eckhardt, D. Arnold, D. Morton, R. L. Thompson, C. D. Groot Zwaaftink, et al. The Lagrangian particle dispersion model FLEXPART version 10.4. *Geoscientific Model Development*, 12(12):4955–4997, 2019.

[19] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[20] M. Reuter, M. Buchwitz, O. Schneising, S. Krautwurst, C. W. O'Dell, A. Richter, H. Bovensmann, and J. P. Burrows. Towards monitoring localized CO_2 emissions from space: co-located regional CO_2 and NO_2 enhancements observed by the OCO-2 and S5P satellites. *Atmospheric Chemistry and Physics*, 19(14):9371–9383, 2019.

[21] C. D. Rodgers. *Inverse methods for atmospheric sounding: theory and practice*, volume 2. World scientific, 2000.

[22] J. M. Stockie. The Mathematics of Atmospheric Dispersion Modeling. *SIAM Review*, 53(2), 2011.

[23] United Nations. Paris agreement. *United Nations Treaty Collection, Chapter XXVII 7. d*, 2015. Adopted 12.12.2015 and in force since 11.04.2016. Published online: https://treaties.un.org/pages/ViewDetails.aspx?src=TREATY&mtdsg_no=XXVII-7-d&chapter=27&clang=_en. Accessed May 5, 2022.

[24] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. Used version 1.7.3.

[25] D. Wunch, P. O. Wennberg, G. Osterman, B. Fisher, B. Naylor, C. M. Roehl, C. O'Dell, L. Mandrake, C. Viatte, M. Kiel, et al. Comparisons of the orbiting carbon observatory-2 (OCO-2) X_{CO_2} measurements with TCCON. *Atmospheric Measurement Techniques*, 10(6):2209–2238, 2017.