Data Assimilation

Theory and Applications

Sebastian Garrido

Supervisor:
Dr. Camilo E. Sarmiento

Mathematics
Universidad del Norte
Barranquilla, Atlantico, Colombia
November 18, 2018

1 Introduction

Correctly modelling physical phenomenon is extremely important as they affect day-to-day operations, e.g. weather has a great influence on traffic. Right now we have the mathematical tools and knowledge to efficiently and accurately predict an wide range of physical occurrences for short windows of time. This is due to the fact that natural systems are usually highly chaotic, and thus, small errors in the initial condition or discrepancies between the model and reality add up such that our predictions tend to diverge from reality after some time. To solve this problem we can use the information from sensors that measure, either directly or indirectly, the system we are making predictions on. Data assimilation is the field which occupies itself with developing techniques that allow integrating information from observations with mathematical models of reality.

In this thesis we will introduce a couple of different data assimilation techniques, most of them based on the Kalman Filter, and we will present two concrete applications of statistical filtering, one in the context of Numerical Weather Predictions, and the other in the context Simultaneous Localization and Mapping.

2 Context

Before giving a rigorous presentation of the Kalman Filter, we would like to give some context on what a filtering problem looks like in reality, and in some way motivate the definitions we will give later on.

Imagine the following situation: We have a mathematical model of wind currents in a particular area. We also have a set of wind speed sensors poorly distributed over the area we are modelling. We guess an initial condition for the model and run it for the next 6 hours. We also record measurements made by the wind speed sensors and compare them with the result of our model. We conclude that our model ran somewhat accurately, but would like to introduce some of the information that our sensors recorded into the model. Unfortunately, the sensors we have do not measure the totality of the area we are modelling, and as such, they present an incomplete observation that cannot be directly inserted into the model as an initial condition. Let's assume, just for this exercise, that they also lack information about which direction the wind is headed to. This means that they are an indirect observation, as they don't capture all the kinds of parameters we may be modelling. In a way, if we could represent this data as vectors, we would obtain that they have non-compatible dimensions. Even though these may seem like unsolvable problems, the Kalman Filter allows us to circumvent these issues.

3 Considerations

From now on, we will assume all vectors are column vectors, except when stated otherwise.

We will also assume that the reader is already well-versed in undergraduate level mathematics and computer science.

4 Classical Kalman Filter

Assume a mathematical model $\mathbf{M}$ of a physical system, a current state of that system $x^b \in \mathbb{R}^m$, an observation $y \in \mathbb{R}^n$, a covariance matrix for the error in the mathematical model $\mathbf{P}^b \in \mathbb{R}^{m \times m}$, a covariance matrix for the error in the observation $\mathbf{R} \in \mathbb{R}^{m \times m}$, and a linear mapping from the space of the model to the space of observations $\mathbf{H} \in \mathbb{R}^{n \times m}$ (also known as the observation operator). If $x^a \in \mathbb{R}^m$ represents the model state after integrating the observations, and $\mathbf{P}^a \in \mathbb{R}^{m \times m}$ represents the model covariance after integrating the observations, then the Kalman Filter equations [17] that allow us to find these values are:

$$x^a = x^b + \mathbf{K}(y - \mathbf{H}x^b)$$
$$\mathbf{P}^a = (\mathbf{I} - \mathbf{K}\mathbf{H})\mathbf{P}^b$$

where $\mathbf{I}$ represents a compatible identity matrix and $\mathbf{K}$, known as the Kalman Gain, is defined as:

$$\mathbf{K} = \mathbf{P}^b\mathbf{H}^T(\mathbf{R} + \mathbf{H}\mathbf{P}^b\mathbf{H}^T)^{-1}$$

Note that the b superscript represents the background, or initial, phase, while the a superscript represents the analysis, or final, phase of the filtering procedure. Also note that the physical system is not used in the Kalman Filter (KF) equations. Presenting it in this section is intentional, as it will become more important later on.

5 Proof of the Classical Kalman Filter

Assume all errors in the following proof are normally distributed with zero mean, that all covariance matrices are symmetrical, and that the observation operator is orthogonal. Assuming that x^* is the real model state, then the model background state and the observation are sampled as:

$$x^b \sim \mathcal{N}(x^*, \mathbf{P}^b)$$
$$y \sim \mathcal{N}(\mathbf{H}x^*, \mathbf{R})$$

Then, maximum likelihood estimation gives us the following expression:

$$\frac{1}{\sqrt{(2\pi)^m|\mathbf{P}^b|}} e^{-\frac{1}{2}(x^b-x^*)^T(\mathbf{P}^b)^{-1}(x^b-x^*)} \cdot \frac{1}{\sqrt{(2\pi)^n|\mathbf{R}|}} e^{-\frac{1}{2}(y-\mathbf{H}x^*)^T\mathbf{R}^{-1}(y-\mathbf{H}x^*)}$$

which can be simplified to:

$$\frac{1}{\sqrt{(2\pi)^{m+n}|\mathbf{P}^b||\mathbf{R}|}} e^{-\frac{1}{2}((x^b-x^*)^T(\mathbf{P}^b)^{-1}(x^b-x^*)+(y-\mathbf{H}x^*)^T\mathbf{R}^{-1}(y-\mathbf{H}x^*))}$$

As we are optimizing with respect to x^* (we need to find the most likely state for our model that could generate both samples), we only need to take into account the variable aspects of the exponent. Therefore, we only need to optimize the following expression:

$$(x^b - x^*)^T(\mathbf{P}^b)^{-1}(x^b - x^*) + (y - \mathbf{H}x^*)^T\mathbf{R}^{-1}(y - \mathbf{H}x^*)$$

To find the optimal value of x^* we derive with respect to it and equate the result to zero. Using matrix-vector derivation properties we arrive at the following expression:

$$
\begin{aligned}
&\Rightarrow 0 = -2\,(x^b - x^*)^T(\mathbf{P}^b)^{-1} - 2\,(y - \mathbf{H}x^*)^T\mathbf{R}^{-1}\mathbf{H} \\
&\Rightarrow 0 = (x^b - x^*)^T(\mathbf{P}^b)^{-1} + (y - \mathbf{H}x^*)^T\mathbf{R}^{-1}\mathbf{H} \\
&\Rightarrow 0 = (\mathbf{P}^b)^{-T}(x^b - x^*) + \mathbf{H}^T\mathbf{R}^{-T}(y - \mathbf{H}x^*) \\
&\Rightarrow 0 = (\mathbf{P}^b)^{-1}(x^b - x^*) + \mathbf{H}^T\mathbf{R}^{-1}(y - \mathbf{H}x^*) \\
&\Rightarrow (\mathbf{P}^b)^{-1}x^* + \mathbf{H}^T\mathbf{R}^{-1}\mathbf{H}x^* = (\mathbf{P}^b)^{-1}x^b + \mathbf{H}^T\mathbf{R}^{-1}y \\
&\Rightarrow ((\mathbf{P}^b)^{-1} + \mathbf{H}^T\mathbf{R}^{-1}\mathbf{H})x^* = (\mathbf{P}^b)^{-1}x^b + \mathbf{H}^T\mathbf{R}^{-1}y \\
&\Rightarrow x^* = ((\mathbf{P}^b)^{-1} + \mathbf{H}^T\mathbf{R}^{-1}\mathbf{H})^{-1}((\mathbf{P}^b)^{-1}x^b + \mathbf{H}^T\mathbf{R}^{-1}y) \\
&\Rightarrow x^* = ((\mathbf{P}^b)^{-1} + \mathbf{H}^T\mathbf{R}^{-1}\mathbf{H})^{-1}((\mathbf{P}^b)^{-1}x^b \\
&\qquad\quad + \mathbf{H}^T\mathbf{R}^{-1}\mathbf{H}x^b - \mathbf{H}^T\mathbf{R}^{-1}\mathbf{H}x^b + \mathbf{H}^T\mathbf{R}^{-1}y) \\
&\Rightarrow x^* = x^b + ((\mathbf{P}^b)^{-1} + \mathbf{H}^T\mathbf{R}^{-1}\mathbf{H})^{-1}(-\mathbf{H}^T\mathbf{R}^{-1}\mathbf{H}x^b + \mathbf{H}^T\mathbf{R}^{-1}y) \\
&\Rightarrow x^* = x^b + ((\mathbf{P}^b)^{-1} + \mathbf{H}^T\mathbf{R}^{-1}\mathbf{H})^{-1}\mathbf{H}^T\mathbf{R}^{-1}(y - \mathbf{H}x^b) \\
&\Rightarrow x^* = x^b + \mathbf{K}^*(y - \mathbf{H}x^b)
\end{aligned}
$$

where $\mathbf{K}^* := ((\mathbf{P}^b)^{-1} + \mathbf{H}^T\mathbf{R}^{-1}\mathbf{H})^{-1}\mathbf{H}^T\mathbf{R}^{-1}$ is known as the information version of the Kalman Gain. This version of the Kalman Gain will become important during the applications as it allows us to set an initial covariance matrix of infinite magnitude (its inverse will just be the zero matrix), which can reduce some of the guesswork involved when estimating initial parameters. This version of the Kalman Gain is also less sensitive to noise. We will now proceed to derive the usual version of the Kalman Gain by proving that $\mathbf{K}^{-1}\mathbf{K}^* = \mathbf{I}$ and therefore $\mathbf{K} = \mathbf{K}^*$. So:

$$
\begin{aligned}
\mathbf{K}^{-1}\mathbf{K}^* &= (\mathbf{H}\mathbf{P}^b\mathbf{H}^T + \mathbf{R})\mathbf{H}(\mathbf{P}^b)^{-1}((\mathbf{P}^b)^{-1} + \mathbf{H}^T\mathbf{R}^{-1}\mathbf{H})^{-1}\mathbf{H}^T\mathbf{R}^{-1} \\
&= (\mathbf{H} + \mathbf{R}\mathbf{H}(\mathbf{P}^b)^{-1})((\mathbf{P}^b)^{-1} + \mathbf{H}^T\mathbf{R}^{-1}\mathbf{H})^{-1}\mathbf{H}^T\mathbf{R}^{-1} \\
&= [\mathbf{R}\mathbf{H}((\mathbf{P}^b)^{-1} + \mathbf{H}^T\mathbf{R}^{-1}\mathbf{H})(\mathbf{H} + \mathbf{R}\mathbf{H}(\mathbf{P}^b)^{-1})^{-1}]^{-1} \\
&= [(\mathbf{R}\mathbf{H}(\mathbf{P}^b)^{-1} + \mathbf{H})(\mathbf{H} + \mathbf{R}\mathbf{H}(\mathbf{P}^b)^{-1})^{-1}]^{-1} \\
&= [(\mathbf{R}\mathbf{H}(\mathbf{P}^b)^{-1} + \mathbf{H})(\mathbf{R}\mathbf{H}(\mathbf{P}^b)^{-1} + \mathbf{H})^{-1}]^{-1} \\
&= \mathbf{I}^{-1} \\
&= \mathbf{I}
\end{aligned}
$$

Let's now derive the covariance matrix for the error in x^*, assuming $\mathbf{R}$, $\mathbf{P}^b$, $\mathbf{H}$, and y are constant:

$$
\begin{aligned}
\mathbf{P}^* &:= Var[x^*] \\
&= Var[x^b + \mathbf{K}(y - \mathbf{H}x^b)] \\
&= Var[x^b] + Var[\mathbf{K}y] - Var[\mathbf{KH}x^b] \\
&= \mathbf{P}^b - \mathbf{KHP}^b \\
&= (\mathbf{I} - \mathbf{KH})\mathbf{P}^b
\end{aligned}
$$

We finish our proof by setting $x^a := x^*$ and $\mathbf{P}^a = \mathbf{P}^*$.

6 Limitations of the Classical Kalman Filter

Clearly, the formulation of the Kalman Filters asks for conditions that are not usually present in practice:

- The observation operator is rarely linear and orthogonal.
- Physical models are rarely linear.
- Model and observation are not always well-behaved.

There are also the memory and computational issues with the Kalman Filter algorithm. Because most realistic physical models have in the order of $O(10^5)$ or $O(10^6)$ data points, storing complete representations of the covariance matrix $\mathbf{P}^b$ and the observation operator $\mathbf{H}$ is usually unfeasible. Even if we ignore the memory problem, we still have to deal with expensive matrix operations over extremely large matrices. In the next few sections we will introduce some techniques to circumvent these issues and make statistical filtering practical.

7 Extended Kalman Filter (EKF)

Given a non-linear mathematical model $\mathcal{M}$, and a non-linear observation operator $\mathcal{H}$, the main idea of the EKF [23] is locally linearize both of them using their first-order approximation. If $\mathbf{J}$ is the jacobian operator, then the EKF reduces both of them, at assimilation time t, to:

$$
\begin{aligned}
\mathbf{M}(x) &= \mathcal{M}(x^a_{t-1}) + (\mathbf{J}\mathcal{M})(x^a_{t-1}) \cdot (x - x^a_{t-1}) \\
\mathbf{H}(x) &= \mathcal{H}(x^a_{t-1}) + (\mathbf{J}\mathcal{H})(x^a_{t-1}) \cdot (x - x^a_{t-1})
\end{aligned}
$$

8 Ensemble Kalman Filter (EnKF)

To solve the representation problem of $\mathbf{P}^b$ we will estimate it using an ensemble of N members sampled from the distribution $\mathcal{N}(x^b, \mathbf{P}^b)$ [17]. We will then apply a modified version of the Kalman Filter over this ensemble, which will produce an analysis ensemble. This new ensemble will allow us to estimate $\mathbf{P}^a$. Specifically, let's assume x_1^b, x_2^b, ..., x_N^b are the members of the background ensemble (the ones estimating $\mathbf{P}^b$). Also assume that y_1, y_2, ..., y_N are a set of disturbed observations, sampled from $\mathcal{N}(y, \mathbf{R})$. Now define the following matrices:

$$\mathbf{X}^b = \begin{pmatrix} x_1^b & x_2^b & \dots & x_N^b \end{pmatrix}$$
$$\mathbf{Y} = \begin{pmatrix} y_1 & y_2 & \dots & y_N \end{pmatrix}$$

Then the equation that allows us to derive the analysis ensemble is:

$$\mathbf{X}^a = \mathbf{X}^b + \mathbf{K}(\mathbf{Y} - \mathbf{H}\mathbf{X}^b)$$

which is structurally identical to the Kalman Filter for the model state. Both $\mathbf{P}^b$ and $\mathbf{P}^a$ are calculated as the covariance matrix of their respective ensemble, where the ensemble members of the analysis ensemble are the columns of $\mathbf{X}^a$.

9 Limitations of the Ensemble Kalman Filter

Because N is usually in the order of $O(10^3)$, it is usually much smaller than m and the estimated covariance matrix is therefore rank-deficient [5]. This prevents inverting the covariance matrix and also brings with it a host of numerical issues. In a later section we will solve this problem using localization. We will also experience convergence of the ensemble members, which may introduce estimation errors as they may become unrepresentative of the distribution they are meant to represent. This issue will be solved by inflation, which again, will be introduced in a later section.

10 Iterative Application of Statistical Filters

Before continuing we would like to present an overview of how to apply the Kalman Filter iteratively. In practice we rarely assimilate once during the lifetime of the model run as we usually run models for months or even years, expecting meaningful predictions every few hours. This is achieved by assimilating

sensor data as soon as we get it. Assume a given interval between assimilations Δ (which may be variable). Assume also that the current timestep is t. If the result of the $t-1$ assimilation is x^a_{t-1} (or X^a_{t-1} if using EnKF) then we can calculate the current background model state using the (not necessarily linear) model $\mathcal{M}$ by $x^b_t = \mathcal{M}(x^a_{t-1})$. If we are using an ensemble of model states, we should propagate each individual member in time to obtain the background matrix for the next timestep. We can then perform the assimilation step to obtain x^a_t. This procedure can then be repeated ad-infinitum. In practice, this procedure also results in the model error decreasing substantially over time. Notice that we have left out of this discussion how to obtain the initial model state. The next section will present a way to estimate the initial model state efficiently. In a later section we will also show how it replaced a previous technique due to its lower computational cost and similar qualitative properties.

11 Ensemble Kalman Smoother (EnKS)

To solve the initial condition problem we will use the Ensemble Kalman Smoother [5]. First we need to guess an initial condition $x^b_0 \in \mathbb{R}^m$ and initial covariance matrix $\mathbf{P}_0 \in \mathbb{R}^{m \times m}$. Then sample a set of ensemble members $x^b_{0,1}$, $x^b_{0,2}$, ... , $x^b_{0,N}$, and propagate each of them t times. This will result in sets of ensemble members $x^b_{k,1}$, $x^b_{k,2}$, ... , $x^b_{k,N}$, for $1 \leq k \leq t$. Define:

$$\mathbf{X}^b = \begin{pmatrix} x^b_{0,1} & x^b_{0,2} & \cdots & x^b_{0,N} \\ x^b_{1,1} & x^b_{1,2} & \cdots & x^b_{1,N} \\ x^b_{2,1} & x^b_{2,2} & \cdots & x^b_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ x^b_{t,1} & x^b_{t,2} & \cdots & x^b_{t,N} \end{pmatrix}$$

Assume also that we have observations y_k, for $0 \leq k \leq t$. For each of them, sample a set of disturbed observations $y_{k,1}$, $y_{k,2}$, ... , $y_{k,N}$ and define:

$$\mathbf{Y} = \begin{pmatrix} y_{0,1} & y_{0,2} & \cdots & y_{0,N} \\ y_{1,1} & y_{1,2} & \cdots & y_{1,N} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ y_{t,1} & y_{t,2} & \cdots & y_{t,N} \end{pmatrix}$$

If $\mathbf{H}_0$, $\mathbf{H}_1$, ... , $\mathbf{H}_t$ are the observation operators for each timestep and $\mathbf{R}_0$, $\mathbf{R}_1$, ... , $\mathbf{R}_t$ are the error covariances for each of the observations then define:

$$\mathbf{H} = \begin{pmatrix} H_0 & 0 & \dots & 0 \\ 0 & H_1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & H_t \end{pmatrix}$$

$$\mathbf{R} = \begin{pmatrix} R_0 & 0 & \dots & 0 \\ 0 & R_1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & R_t \end{pmatrix}$$

Then calculate $\mathbf{X}^a$ using the EnKF equations. Finally, the optimal initial condition is the mean of the column vectors of the submatrix $\mathbf{S} := (\mathbf{X}^a_{i,j})^{i=1,2,\dots,m}_{j=1,2,\dots,N}$.

Because the EnKS uses the same equations as the EnKF, it is also possible to reuse existing implementations of the latter when using it. This makes EnKS extremely practical.

12 Localization

When estimating $\mathbf{P}^b$ from its respective ensemble members, we might obtain spurious correlations between physically unrelated data points in the model due to both numerical issues and the rank-deficiency of the ensemble. For example, assume the following set of differential equations:

$$\frac{\partial p_i}{\partial t}(t) = p_{(i-1 \bmod 40)}(t) + p_{(i+1 \bmod 40)}(t),\ i = 0, 1, ..., 39$$

Because each point is only affected by its neighbours we expect the covariance matrix associated with such model to be banded for a small enough Δ, i.e. to be zero everywhere except in a small band around the main diagonal. In reality, because of the different types of error we describe earlier, the matrix we actually obtain looks somewhat like this:

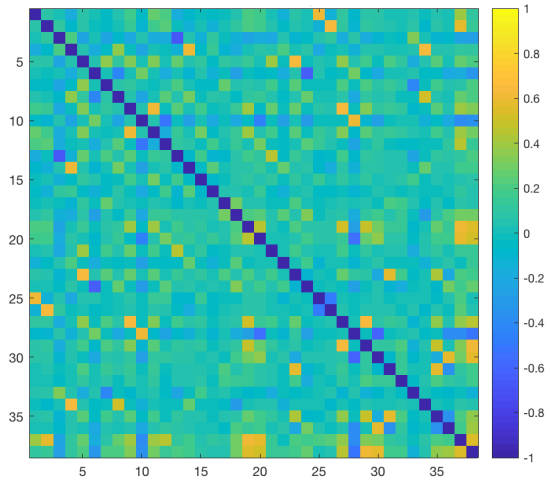

Figure 1: Model error covariance before localization [3]

One way to overcome this issue is by using localization. Assume that $\mathbf{P}_l^b$ represents the covariance matrix after localization. Then this matrix can be computed using the Schur (or element-wise) product:

$$\mathbf{P}_l^b = \rho \circ \mathbf{P}^b$$

where ρ can be any compatible matrix, but is usually [5]:

$$(\rho_{0,1})_{i,j} := \begin{cases} 1 & \text{if } d(i,j) \leq d_{\max} \\ 0 & \text{otherwise} \end{cases}$$

for some $d_{\max} \in \mathbb{N}$. In general, ρ should have the form $(\rho)_{i,j} := c(i,j)$, where c is a moderation function that satisfies $c(i,j) = 1$ when $i = j$ and $c(i,j) \to 0$ when $|i-j| \to \infty$. The intuition behind these choices is that elements that are separated by enough distance should have a tenuous-at-best relationship. Mathematically, they allow artificially increasing the rank of our covariance matrices.

Continuing with our example, the application of the $\rho_{0,1}$ localization matrix to our example covariance results in:

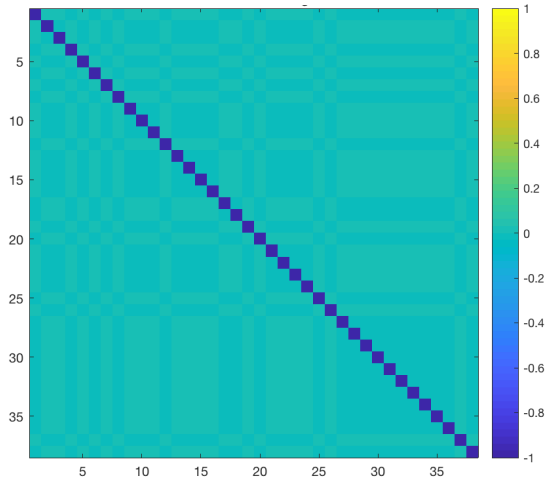

Figure 2: Model error covariance after localization [3]

From this example it is clear that $\rho_{0,1}$ is of special interest as it zeroes out elements outside of a certain radius. This allows storing the resulting matrix in a sparse format, which is both memory and computationally efficient. Also, because the resulting matrix is banded we can afford extra optimizations.

13 Inflation

Iterative application of the filtering algorithm sooner or later produces convergence of our ensemble members. This is undesirable as it is likely that the mean and the covariance matrix associated with this new ensemble will not be representative of the statistical properties of the underlying model error. To solve this problem there are several popular approaches: resampling the ensemble with increased spread, additive/multiplicative inflation [15], or square-root filter inflation. The first is self-explanatory, with the inflation parameter obtained experimentally and varying from situation to situation. The second we will not cover due to scope. The final one will be covered in the next section as we cover the square-root type filters.

14 Square Root Filters

It is common knowledge that population covariance matrices are positive-definite. As such, $\mathbf{P}^a$ ought to be positive-definite. Unfortunately, due to numerical instability in the Kalman Filter, small amounts of noise in the input may affect the sign of the eigenvalues, which would prevent $\mathbf{P}^a$ from being positive-definite after the analysis stage.

One solution to this problem is representing $\mathbf{P}^b$ in a way that allows manipulating it without losing the positive-definite property. Using the Cholesky Decomposition helps us obtain the square root $\mathbf{S}^b$ [24] of this matrix (which satisfies $\mathbf{P}^b = \mathbf{S}^b(\mathbf{S}^b)^T$). We can now replace $\mathbf{S}^b$ in the Kalman Filter equations and obtain a set of expressions that allows us to obtain the analysis model state x^a and model covariance $\mathbf{S}^a$ in terms of $\mathbf{S}^b$. Finally we can reconstruct $\mathbf{P}^a$ as $\mathbf{S}^a(\mathbf{S}^a)^T$.

Another type of square root filter that we will present in a later section is the Local Ensemble Transform Kalman Filter. It uses many of the techniques we have already described which makes it one of the most accurate and efficient data assimilation algorithms to date. It is currently in use in several Numerical Weather Prediction (NWP) centers, as it improved upon the state-of-the-art assimilation quality and computational cost. We will now present some numerical concepts and techniques before introducing this algorithm.

15 Numerical Concepts and Techniques

15.1 Spurious Eigenvalues

Due to numerical factors (approximation, instability, etc.) it is possible finding relatively small eigenvalues that are not related in any way to the physical phenomenon that is being modelled. It is therefore desirable to be able to cancel out or remove these eigenvalues from a matrix as they are mostly noise. SVD Decomposition is one tool that allows us to do just this.

15.2 SVD Decomposition

Let $\mathbf{A}$ be a real, symmetric, square matrix. Then there are real, orthogonal matrices $\mathbf{U}$ and $\mathbf{V}$, and a diagonal matrix $\mathbf{\Sigma}$ with real non-negative entries such that $\mathbf{A} = \mathbf{U\Sigma V}^*$ [1]. Moreover, the entries of $\mathbf{\Sigma}$ are the eigenvalues of $\mathbf{A}$. They are also usually written from largest to smallest along the principal diagonal.

This type of decomposition is extremely useful as it allows us to both observe and directly manipulate the eigenvalues of a matrix. In particular, we will use this technique to remove spurious eigenvalues from the pseudoinverse of a

matrix.

It is also important to note that the SVD Decomposition can be taken of arbitrary real or complex matrices, but that in the general case the values in the diagonal of $\mathbf{\Sigma}$ are not the eigenvalues but instead the singular values of the matrix $\mathbf{A}$.

15.3 Linear Least Squares

Given a matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$, and a vector $b \in \mathbb{R}^n$. We would like to find the value of $x \in \mathbb{R}^m$ that minimizes the expression $\|\mathbf{A}x - b\|$. Clearly, when $\mathbf{A}$ is invertible the solution is $x = \mathbf{A}^{-1}b$. In the more general case it is also possible to find what is known as the pseudoinverse of $\mathbf{A}$, denoted $\mathbf{A}^+$, such that $\mathbf{A}^+b$ solves the Linear Least Squares problem. In the next subsection we will prove both the existence and unicity of the Moore-Penrose Pseudoinverse, derive its equation and show that it solves the linear least squares problem.

15.4 Pseudoinverse

The Moore-Penrose Pseudoinverse [1] of $\mathbf{A} \in \mathbb{R}^{n \times m}$ is defined as the matrix $\mathbf{A}^+ \in \mathbb{R}^{m \times n}$ that satisfies:

- $\mathbf{A}\mathbf{A}^+\mathbf{A} = \mathbf{A}$
- $\mathbf{A}^+\mathbf{A}\mathbf{A}^+ = \mathbf{A}^+$
- $(\mathbf{A}\mathbf{A}^+)^* = \mathbf{A}\mathbf{A}^+$
- $(\mathbf{A}^+\mathbf{A})^* = \mathbf{A}^+\mathbf{A}$

Let's show that this pseudoinverse is unique. Suppose that for a given matrix $\mathbf{A}$ there are two matrices $\mathbf{B}$ and $\mathbf{C}$ satisfying the required properties. Note that:

$$\mathbf{AB} = \mathbf{ACAB} = (\mathbf{AC})^*(\mathbf{AB})^* = \mathbf{C}^*\mathbf{A}^*\mathbf{B}^*\mathbf{A}^* = \mathbf{C}^*\mathbf{A}^* = (\mathbf{AC})^* = \mathbf{AC}$$

Analogously, $\mathbf{BA} = \mathbf{CA}$. Then:

$$\mathbf{B} = \mathbf{BAB} = \mathbf{BAC} = \mathbf{CAC} = \mathbf{C}$$

Therefore the Moore-Penrose Pseudoinverse is unique. To show existence let $\mathbf{A}^+ = (\mathbf{A}^*\mathbf{A})^{-1}\mathbf{A}^*$. Then:

- $\mathbf{A}\mathbf{A}^{+}\mathbf{A} = \mathbf{A}(\mathbf{A}^{*}\mathbf{A})^{-1}\mathbf{A}^{*}\mathbf{A} = \mathbf{A}$
- $\mathbf{A}^{+}\mathbf{A}\mathbf{A}^{+} = (\mathbf{A}^{*}\mathbf{A})^{-1}\mathbf{A}^{*}\mathbf{A}\mathbf{A}^{+} = \mathbf{A}^{+}$
- $(\mathbf{A}\mathbf{A}^{+})^{*} = (\mathbf{A}(\mathbf{A}^{*}\mathbf{A})^{-1}\mathbf{A}^{*})^{*} = (\mathbf{A}\mathbf{A}^{-1}(\mathbf{A}^{*})^{-1}\mathbf{A}^{*})^{*} = \mathbf{I}^{*} = \mathbf{I} = \mathbf{A}\mathbf{A}^{+}$
- $(\mathbf{A}^{+}\mathbf{A})^{*} = ((\mathbf{A}^{*}\mathbf{A})^{-1}\mathbf{A}^{*}\mathbf{A})^{*} = \mathbf{I}^{*} = \mathbf{I} = (\mathbf{A}^{*}\mathbf{A})^{-1}\mathbf{A}^{*}\mathbf{A} = \mathbf{A}^{+}\mathbf{A}$

which proves existence. To show that $\mathbf{A}^{+}$ solves the the Linear Least Squares problem, let $z = \mathbf{A}^{+}b$ and let $x \in \mathbb{R}^{m}$. Then:

$$\begin{aligned}
\|\mathbf{A}x - b\|^{2} &= \|\mathbf{A}x - b - \mathbf{A}z + \mathbf{A}z\|^{2} \\
&= \|\mathbf{A}(x - z) + \mathbf{A}z - b\|^{2} \\
&= \|\mathbf{A}(x - z)\|^{2} + (\mathbf{A}(x - z))^{*}(\mathbf{A}z - b) \\
&\quad + ((\mathbf{A}(x - z))^{*}(\mathbf{A}z - b))^{*} + \|\mathbf{A}z - b\|^{2} \\
&= \|\mathbf{A}(x - z)\|^{2} + (x - z)^{*}\mathbf{A}^{*}(\mathbf{A}z - b) \\
&\quad + ((x - z)^{*}\mathbf{A}^{*}(\mathbf{A}z - b))^{*} + \|\mathbf{A}z - b\|^{2}
\end{aligned}$$

but:

$$\begin{aligned}
\mathbf{A}^{*}(\mathbf{A}z - b) &= \mathbf{A}^{*}(\mathbf{A}\mathbf{A}^{+}b - b) \\
&= \mathbf{A}^{*}\mathbf{A}(\mathbf{A}^{*}\mathbf{A})^{-1}\mathbf{A}^{*}b - \mathbf{A}^{*}b \\
&= \mathbf{A}^{*}b - \mathbf{A}^{*}b \\
&= 0
\end{aligned}$$

so:

$$\begin{aligned}
\|\mathbf{A}x - b\|^{2} &= \|\mathbf{A}(x - z)\|^{2} + \|\mathbf{A}z - b\|^{2} \\
&\geq \|\mathbf{A}z - b\|^{2}
\end{aligned}$$

which proves that the Moore-Penrose Pseudoinverse is a solution to the Linear Least Squares problem. Finally, let's find an expression for the pseudoinverse in terms of the result of SVD decomposition. This will give us the opportunity to manipulate the eigenvalues of a matrix midway through the pseudoinversion process, something which will become useful during the applications.

So, if $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{*}$, then:

$$
\begin{aligned}
\mathbf{A}^{+} &= (\mathbf{A}^{*}\mathbf{A})^{-1}\mathbf{A}^{*} \\
&= (\mathbf{V}\mathbf{\Sigma}^{*}\mathbf{U}^{*}\mathbf{U}\mathbf{\Sigma}\mathbf{V}^{*})^{-1}\mathbf{V}\mathbf{\Sigma}^{*}\mathbf{U}^{*} \\
&= (\mathbf{V}\mathbf{\Sigma}^{*}\mathbf{\Sigma}\mathbf{V}^{*})^{-1}\mathbf{V}\mathbf{\Sigma}^{*}\mathbf{U}^{*} \\
&= (\mathbf{V}\mathbf{\Sigma}\mathbf{\Sigma}\mathbf{V}^{*})^{-1}\mathbf{V}\mathbf{\Sigma}\mathbf{U}^{*} \\
&= (\mathbf{V}\mathbf{\Sigma}^{2}\mathbf{V}^{*})^{-1}\mathbf{V}\mathbf{\Sigma}\mathbf{U}^{*} \\
&= (\mathbf{V}^{*})^{-1}\mathbf{\Sigma}^{-2}\mathbf{V}^{-1}\mathbf{V}\mathbf{\Sigma}\mathbf{U}^{*} \\
&= (\mathbf{V}^{*})^{-1}\mathbf{\Sigma}^{-2}\mathbf{\Sigma}\mathbf{U}^{*} \\
&= \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^{*}
\end{aligned}
$$

15.5 Domain Decomposition

Suppose the model state represents a certain quantity over a $k_1 \times k_2 \times \ ... \ \times k_s$ s-dimensional grid. If we are using $\rho_{0,1}$ localization with threshold $d_{\max}$, we can subdivide the domain into a large number of small grids. Notice that all elements in these small subgrids depend only on themselves plus a small padding around the grid whose size depends on $d_{\max}$. Because assimilating over these small windows (with a properly-chosen window size) is exponentially faster than assimilating over the full model state we can substantially improve the assimilation speed of our algorithm by performing the assimilation stage over these smaller windows and reconstructing the full analysis by stitching together the results. This also allows massive parallelization of the assimilation algorithm, as each procedure can run fully independent from each other.

16 Local Ensemble Transform Kalman Filter

16.1 Description

The Local Ensemble Transform Kalman Filter (LETKF) [16] is an efficient and stable square root filter, currently in use in many operating NWP centers to perform meteorological data assimilation. Using the square root ensures that we obtain a proper sample mean, that the dependence between members of the ensemble is continuous and thus physically related data points do not diverge too much from each other, and finally, to ensure that analysis perturbations are not so different from background perturbations.

16.2 Aberration Matrix

An abberation matrix for a set of column vectors v_1, v_2, ... , v_N is the matrix:

$$\begin{pmatrix} v_1 & v_2 & \dots & v_N \end{pmatrix} - \begin{pmatrix} \bar{v} & \bar{v} & \dots & \bar{v} \end{pmatrix}$$

where $\bar{v}$ is the mean of the collection of vectors $(v_i)_{i=1,2,\dots,N}$.

16.3 Algorithm

Assume an ensemble x_1^b, x_2^b, ... , x_N^b sampled with covariance $\mathbf{P}^b$. Assume also an observation y with covariance $\mathbf{R}$ and an observation operator $\mathbf{H}$. Assume also an inflation parameter σ. Finally, assume a $\rho_{0,1}$ localization matrix with a given threshold $d_{\max}$. Then the LETKF algorithm is as follows:

- Let $\bar{y}^b$ be the mean of the collection of vectors $(\mathbf{H}x_i^b)_{i=1,2,\dots,N}$ and $\mathbf{Y}^b$ be its aberration matrix.
- Let $\bar{x}^b$ be the mean of the collection of vectors $(x_i^b)_{i=1,2,\dots,N}$ and $\mathbf{X}^b$ be its aberration matrix.
- For each data point in the model state do the following:
- Let i be the current index in the data model.
- Let $I = \{j \mid \rho_{0,1}(i,j) = 1\}$.
- Let $\mathbf{X}_I^a$ be the submatrix of $\mathbf{X}^a$ which has only the rows indexed by I.
- Let $\mathbf{X}_I^b$ be the submatrix of $\mathbf{X}^b$ which has only the rows indexed by I.
- Let $\mathbf{Y}_I^b$ be the submatrix of $\mathbf{Y}^b$ which has only the rows indexed by I.
- Let $\mathbf{R}_I$ be the submatrix of $\mathbf{R}$ which has only the rows indexed by I.
- Let $\bar{x}_I^b$ be the subvector of $\bar{x}^b$ which has only the rows indexed by I.
- Let $\bar{y}_I^b$ be the subvector of $\bar{y}^b$ which has only the rows indexed by I.
- Let y_I^b be the subvector of y^b which has only the rows indexed by I.
- Let $C = (\mathbf{Y}_I^b)^T \mathbf{R}_I^{-1}$.
- Let $\tilde{\mathbf{P}^a} = [(N-1)\mathbf{I}/\sigma + C\mathbf{Y}_I^b]^{-1}$, where $\mathbf{I}$ is a compatible identity matrix.
- Let $W_1^a = [(N-1)\tilde{\mathbf{P}^a}]^{\frac{1}{2}}$.
- Let $w^a = \tilde{\mathbf{P}^a}(y_I - \bar{y}_I^b)$.
- Let $W_2^a = W_1^a + (\ w^a\ w^a\ \dots\ w^a\ )$.
- Finally set $X_I^a = \bar{x}_I^b + X_I^b W_2^a$ and repeat the previous steps with the next row index. If there are no more rows, then the algorithm has finished executing. Notice that this way we are constructing X^a row-by-row, which is a type of domain decomposition.

17 Application: State-of-the-art Meteorological Data Assimilation

17.1 Introduction

In this section we will introduce a previous state-of-the-art production setup for meteorological data assimilation, show its shortcomings and present a modern setup that addresses these practical issues. Before continuing we must introduce some ideas to understand why the previous setup is less desirable than the current one.

17.2 Automatic Differentiation (AD)

One often-ignored aspect of differential calculus is that function evaluation and derivative evaluation can be performed side-to-side [20]. In computational mathematics we are often taught to approximate the derivative using secants. In fact, we can compute the derivative exactly by following a set of rules that are very natural for anyone familiar with derivatives. To do this, we will replace the x value of the function with a tuple (x, x'), that follows the following rules:

- $\text{constant}_c(x, x') = (c, 0)$
- $(x_1, x_1') + (x_2, x_2') = (x_1 + x_2, x_1' + x_2')$
- $(x_1, x_1') - (x_2, x_2') = (x_1 - x_2, x_1' - x_2')$
- $(x_1, x_1') \cdot (x_2, x_2') = (x_1 x_2, x_1' x_2 + x_1 x_2')$
- $(x_1, x_1')/(x_2, x_2') = (\frac{x_1}{x_2}, \frac{x_1' x_2 - x_1 x_2'}{x_2^2})$
- $(x, x')^n = (x^n, n(x')^{n-1})$
- $\sin(x, x') = (\sin(x), x'\cos(x))$
- $\cos(x, x') = (\cos(x), -x'\sin(x))$
- $\exp(x, x') = (\exp(x), x'\exp(x))$
- $\ln(x, x') = (\ln(x), \frac{x'}{x})$
- $|(x, x')| = (|x|, x'\text{sign}(x))$

In particular, for an initial value of x_0, the initial tuple should be $(x_0, 1)$. After simplifying the left side of the tuple will be the function value and the right side will be the value of the derivative at that point.

To exemplify let $f(x) = x(e^x + (x+2)^2)$. Clearly, $f'(x) = (e^x + (x+2)^2) + x(e^x + 2(x+2))$. If $x = 0$ then $f(0) = 0$ and $f'(0) = 5$. Let's prove this:

$$
\begin{aligned}
f(0,1) &= (0,1) \cdot (\exp(0,1) + ((0,1) + (2,0))^2) \\
&= (0,1) \cdot (\exp(0,1) + (2,1)^2) \\
&= (0,1) \cdot (\exp(0,1) + (4,2)) \\
&= (0,1) \cdot ((1,1) + (4,2)) \\
&= (0,1) \cdot (5,3) \\
&= (0,5)
\end{aligned}
$$

17.3 Tangent Linear Model and Adjoint Model

For a linear model $\mathbf{M}$ its Tangent Linear Model [6] at time t_i is $\mathbf{M'} = \frac{\partial \mathbf{M}[x(t_i)]}{\partial x}$. Its Perturbation Model (to propagate perturbations in time) is $\delta x(t_{i+1}) = \mathbf{M'}\delta x(t_i)$. The Adjoint Model is defined as $\mathbf{M}^* = (\mathbf{M'})^T$. Let's calculate the Adjoint Model for an example. Assume the following model:

$$z = x + y^2$$

then the Tangent Linear Model is:

$$\begin{pmatrix} \delta x \\ \delta y \\ \delta z \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 2z & 0 \end{pmatrix} \begin{pmatrix} \delta x \\ \delta y \\ \delta z \end{pmatrix}$$

and the Adjoint Model is:

$$\begin{pmatrix} \delta x^* \\ \delta y^* \\ \delta z^* \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 2z \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \delta x^* \\ \delta y^* \\ \delta z^* \end{pmatrix}$$

We will now prove the correctness of performing a backwards integration using the adjoint model. This will constitute the main step of the 4D-Var algorithm, which allows us to find an optimal initial condition for our model. Assume a guess for the initial condition of the model x_0 and the vectors x_1, x_2, ... , x_t representing the next t timesteps of our model run. Assume also a background model state x^b, and that for each timestep $0 \leq k \leq t$ we have an observation y_k, an observation operator $\mathbf{H}_k$, and a covariance matrix for the

observation $\mathbf{R}_k$. Finally, assume an initial background model covariance $\mathbf{P}_0^b$. Now define the following cost function:

$$J(x_0) = \frac{1}{2}(x_0 - x^b)^T(\mathbf{P}_0^b)^{-1}(x_0 - x^b) + \frac{1}{2}\sum_{k=0}^{t}(\mathbf{H}_k x_k - y_k)^T\mathbf{R}_k^{-1}(\mathbf{H}_k x_k - y_k)$$

Deriving J with respect to the initial condition we get:

$$\nabla J(x_0) = (\mathbf{P}_0^b)^{-1}(x_0 - x^b) + \sum_{k=0}^{t}(\mathbf{M}')_{k\to 0}^T(\mathbf{H}_k')^T\mathbf{R}_k^{-1}(\mathbf{H}_k' x_k - y_k)$$

The main idea now is to study how perturbations in the initial condition affect the cost function. Let $d_k = y_k - \mathbf{H}_k x_k$ and $\delta x_k = \mathbf{M}'\delta x_{k-1}$ with $\delta x_0 = x_0 - x^b$, for $0 \leq k \leq t$. Substituting into $\nabla J(x_0)$ and using the definition of adjoint model:

$$\nabla J(\delta x_0) = (\mathbf{P}_0^b)^{-1}\delta x_0 + \sum_{k=0}^{t}\mathbf{M}_{k\to 0}^*\mathbf{H}_k^*\mathbf{R}_k^{-1}(\mathbf{H}_k\delta x_k - d_k)$$

or with $\delta x_0 = 0$:

$$\nabla J(\delta x_0) = -\sum_{k=0}^{t}\mathbf{M}_{k\to 0}^*\mathbf{H}_k^*\mathbf{R}_k^{-1}d_k$$

Now let's choose the solution of the adjoint system as follows:

$$\begin{aligned} x_{t+1}^* &= 0 \\ x_0^* &= \mathbf{M}_1^* x_1^* \\ x_k^* &= \mathbf{M}_{k+1}^* x_{k+1}^* + \mathbf{H}_k^*\mathbf{R}_k^{-1}d_k,\ 1 \leq k \leq t \end{aligned}$$

Progressive substitution on x_0^* results in:

$$
\begin{aligned}
x_0^* &= \mathbf{M}_1^* x_1^* \\
&= \mathbf{M}_1^*(\mathbf{M}_2^* x_2^* + \mathbf{H}_1^* \mathbf{R}_1^{-1} d_1) \\
&= \mathbf{M}_1^*(\mathbf{M}_2^*(\mathbf{M}_3^* x_3^* + \mathbf{H}_2^* \mathbf{R}_2^{-1} d_2) + \mathbf{H}_1^* \mathbf{R}_1^{-1} d_1) \\
&= \dots \\
&= \sum_{k=0}^{t} \mathbf{M}_1^* \mathbf{M}_2^* \dots \mathbf{M}_k^* \mathbf{H}_k^* \mathbf{R}_k^{-1} d_k \\
&= \sum_{k=0}^{t} \mathbf{M}_{k\to 0}^* \mathbf{H}_k^* \mathbf{R}_k^{-1} d_k \\
&= -\nabla J(\delta x_0)
\end{aligned}
$$

One idea underlying this proof is that of the interior product. If $< \cdot, \cdot >$ is an interior product, then $< \mathbf{M} x_{t-1}, x_t > = < x_{t-1}, \mathbf{M}^* x_t >$. This implies that in some way the adjoint is integrating the model state back in time, at least when looked through the lens of least-square minimization.

17.4 4D-Var

The main idea behind this technique is that studying the sensitivity of a model to perturbations in its parameters (initial condition, boundary condition, etc.) allows us to obtain crucial information, like optimal parameters for any given model run [6]. In particular, 4D-Var tries to obtain an optimal initial condition x_0 to the model with respect to a set of observations and the results of a model propagation. Empirically, it produces very good results, which has led this technique to be implemented in ECMWF, Meteo France, the Met Office, JMA and CMC, among other NWP centers.

To execute 4D-Var perform the following steps:

- Integrate the model forward from the current initial condition. This will allow us to calculate J.
- Integrate the adjoint model back in time to obtain ∇J.
- If ∇J is lower than the tolerance, then our current initial condition is good enough.
- Otherwise, use ∇J to estimate a better initial condition using the gradient descent algorithm.
- If another integration loop is feasible, repeat all the previous steps using the result of the gradient descent algorithm.

17.5 4D-Var Limitations

Notice that there are several limitations with this approach. First the Tangent Linear Model is obtained by deriving the model and can therefore be calculated using automatic differentiation. In fact, for real-life models with thousands of lines of code this may be the only practical way to find the Tangent Linear Model without risking human error during the derivation. Unfortunately, AD has issues when applied to real-life models. The first one is that it cannot deal with non-differentiable instructions like if statements. AD implementations also tend to generate redundant or memory-inefficient statements which must be cleaned by hand [13].

There is also the issue that the Tangent Linear Model is at least at expensive to run as the original model, and usually it is 1.5 as expensive. Due to this high cost the optimization loop can rarely be ran more than once or twice, depending on the computational power of the NWP center performing the computation. Another issue is that the model error covariance is assumed to be constant in time, which is rarely the case in practice. The final issue is that the model covariance error is usually too large to represent in memory, as we've stated earlier.

In the next section we will present a complete production-ready setup that solves these limitations.

17.6 4D-LETKF

Given a guess for the initial model state we can efficiently compute an initial state by setting up our integration parameters just like in the EnKS and feeding them into the LETKF. Notice that because the only innovation in the EnKS is in parameter setup, any ensemble filtering algorithm can be turned into a smoother. Given the optimal initial condition we can then use the traditional LETKF to perform iterative filtering on the incoming sensor data. It might also be numerically useful to use the SVD pseudoinverse whenever inversion is non-trivial.

Because this method doesn't use the Adjoint Model, we can save ourselves the trouble of coding and computing the model derivative which makes 4D-LETKF substantially more efficient, computation-wise, with respect to 4D-Var. Because 4D-LETKF is an ensemble method, we have also solved the representation problem for P^b. Finally, because we can compute P^b from the ensemble members at each timestep, we have solved the model error covariance matrix staticity issue. The only remaining step is to empirically prove the superiority of 4D-LETKF over 4D-Var.

17.7 SPEEDY

SPEEDY [19] (Simplified Parameterizations, privitivE-Equation DYnamics) is a physical weather model that allows us to simulate both local and global phenomenon. It uses an 8-layered data grid of dimensions 96×48 which keeps track of vorticity, specific humidity, divergence, absolute temperature, and the natural logarithm of surface pressure (About 184320 data points). It is also extremely efficient, allowing us to run a full-year simulation in 12 minutes using a modest single-core 3.4 GHz computer.

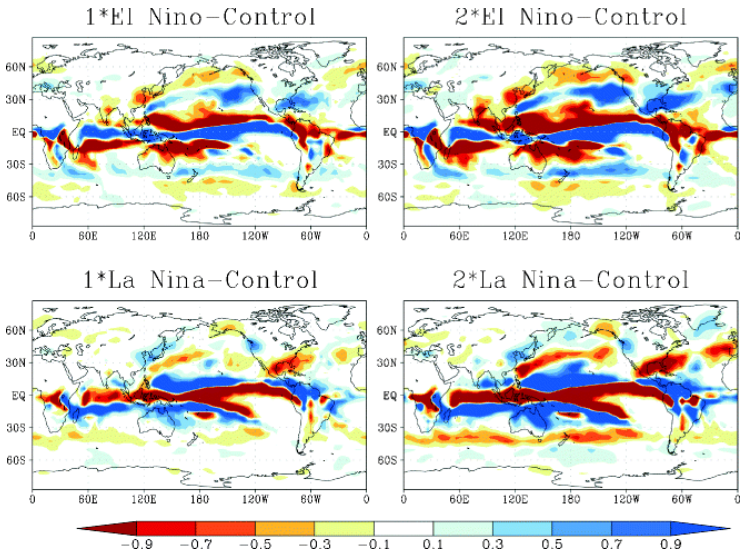

Figure 3: SPEEDY model run [11]

17.8 Experimental Results

The following graph compares 4D-LETKF with 4D-Var performance during data assimilation in production-like environments. The model used for this experiment was the SPEEDY model we introduced in the previous section. The results clearly show that LETKF is a substantial improvement over 4D-Var in terms of computation time, which adds value to the qualitative improvements we already detailed in the previous subsection.

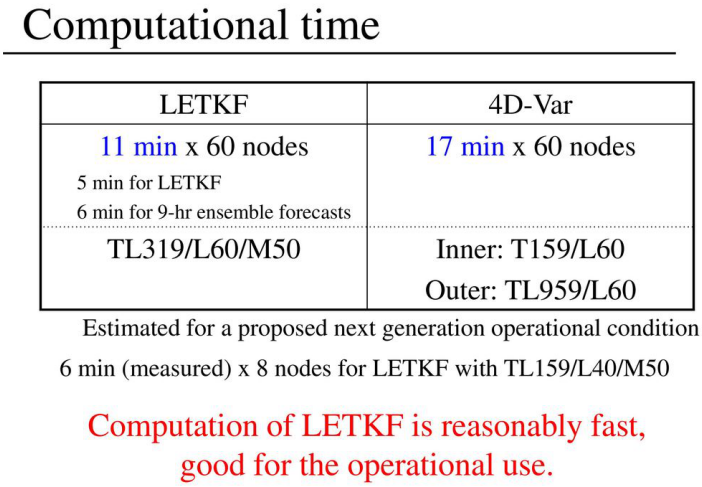

Figure 4: Benchmark results [18]

18 Application: Realtime Simultaneous Localization and Mapping using Monocular Video Sensors

18.1 Introduction

The Simultaneous Localization and Mapping (SLAM) problem is formulated as the problem of determining, given an agent and a set of sensors located at the same position as the agent, the following items:

- Reconstruction of the map surrounding the agent (Mapping Problem)
- Locating the agent in the map that is being drawn (Localization Problem)
- Determining whether a particular location has been visited before by the agent (Loop Closure Problem)

There are many applications of SLAM already in consumer electronics. Some of them include self-driving vehicles (e.g. Tesla), autonomous vacuum cleaner (e.g. Roomba) and tracking/recording drones. These applications are usually fed by a wide array of sensors which might include, but are not necessarily limited to, any of the following:

- Monocular Video

- Stereo Video
- Gyroscope
- Accelerometer
- Magnetometer
- LIDAR
- Motor Feedback

Note that all of these sensors, but the first two, provide information about the current position or movement characteristics of the agent in a straightforward fashion. In particular, our application uses only monocular video to perform scaleless SLAM. To add a sense of scale, stereo vision is needed, as the relative position of the cameras with respect to each other encodes all the necessary information to measure objects inside their frame.

In this section we will introduce MonoSLAM, the first realtime monocular vision algorithm that solves the SLAM problem. In our exposition we will focus only on the Localization and Mapping problems, even though MonoSLAM solves all of the three problems that make up SLAM. We will now introduce some necessary background:

18.2 Video-based SLAM approaches

There are two main approaches when it comes to video-based SLAM: sparse and dense strategies. The former attempt to extract meaningful point-features from the image, while the latter attempt to process the full image, obtaining information from areas or patches of the image. Modern dense approaches include LSD-SLAM and DTAM, while moden sparse approaches include Orb-SLAM (v1 and v2), PTAM and MonoSLAM.

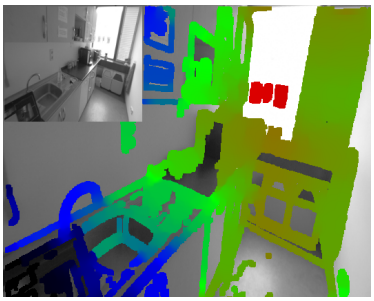

Figure 5: Dense image processing in LSD-SLAM [12]

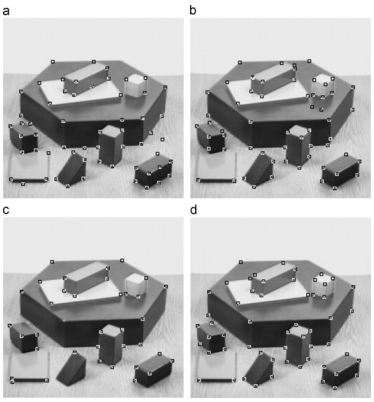

Figure 6: Sparse image processing with Corner Detection [25]

We will now lay the necessary groundwork behind the video processing capabilities of MonoSLAM.

18.3 Pinhole Camera Model

Let $(x, y, z) \in \mathbb{R}^3$. Assume a camera with principal point (u_0, v_0), focal length in x, α_x, focal length in y, α_y, and skew coefficient γ. Then the matrix K of intrinsic camera parameters is:

$$K = \begin{pmatrix} \alpha_x & \gamma & u_0 & 0 \\ 0 & \alpha_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Then, given a rotation matrix R and an origin for the world coordinates T, we can map this 3D point into 2D space using the pinhole camera model [14]:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = K[R \;\; T] \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

18.4 Wide Angle Camera Model

In the case of a wide angle camera with radial distortion θ, we can define an invertible transformation [22] that takes into account this distortion. The equations for the wide angle camera model are:

$$u_d = u_0 + \frac{u - u_0}{\sqrt{1 + 2\theta r^2}}$$
$$v_d = v_0 + \frac{v - v_0}{\sqrt{1 + 2\theta r^2}}$$

where $r = \sqrt{(u - u_0)^2 + (v - v_0)^2}$. Wide angle cameras will become particularly useful during MonoSLAM as they have shown better SLAM results experimentally than traditional pinhole cameras.

18.5 Kernel Convolution

Given a square matrix with odd side-length $\mathbf{G}$ known as the kernel and an arbitrary matrix $\mathbf{A}$, the convolution of $\mathbf{G}$ on $\mathbf{A}$ [2], noted $\mathbf{G} * \mathbf{A}$ is a matrix with dimension equal to that of $\mathbf{A}$, and whose $(\mathbf{G} * \mathbf{A})_{i,j}$ entry is calculated in the following manner:

- Let $\mathbf{Q}$ be a window of $\mathbf{A}$ centered on (i, j), with same size as $\mathbf{G}$. For each cell in the window, if the cell extends outside $\mathbf{A}$, set that cell to zero, otherwise set it to the value of $\mathbf{A}$ at that location.
- Let $\mathbf{Q}' = \mathbf{G} \circ \mathbf{Q}$.
- Finally, $(\mathbf{G} * \mathbf{A})_{i,j} = \sum_{i,j} \mathbf{Q}'_{i,j}$

18.6 Sobel Operator

It would be useful to find the "differential of an image", i.e. a measure of how fast an image changes in each pixel. This would allow us to find edges in the image by looking for areas with high rates of change. The Sobel Operator [2] is one possible option to achieve this. Consider the following two matrices:

$$\mathbf{G}_x = \begin{pmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{pmatrix}$$

$$\mathbf{G}_y = \begin{pmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

Notice that when we apply these matrices using the matrix convolution we defined previously, their form allows us to extract information from the

neighbourhood surrounding a pixel. In particular, $\mathbf{G}_x$ computes a measure of change in the x direction, favoring change near the center point while completely ignoring change in the y direction. $\mathbf{G}_y$ behaves analogously but in the y direction instead. We could potentially combine both matrix convolutions to obtain a measure total change:

$$\mathbf{G} = \sqrt{(\mathbf{G}_x * \mathbf{A})^2 + (\mathbf{G}_y * \mathbf{A})^2}$$

This concept of image differential is particularly important when trying to solve the problem of edge detection. Look for example at the application of the Sobel operator to an image:

Figure 7: Sobel operator applied to an Edge Detection Problem

It deserves to be noted that images are not matrices, as each pixel in an image has associated with it an RGB value. Because of this we must transform the image to grayscale, which can be represented by a matrix, before calculating the Sobel Operator.

18.7 Shi-Tomasi features

Edges and specially corners are good feature to track as they tend to change very little from frame to frame during video streams. Shi and Tomasi [21], following on this idea, derived a feature class from first principles with the intent of making it a good feature to track. This feature is known as a Shi-Tomasi feature. It is computed from an image differential by segmenting the image into windows of small size (11×11, 13×13, and 15×15 have shown to work well in practice). For each window they compute the Shi-Tomasi operator:

$$Z = \begin{pmatrix} g_x^2 & g_x g_y \\ g_x g_y & g_y^2 \end{pmatrix}$$

where

$$g_x^2 = \sum_{i,j} (\mathbf{G}_x * \mathbf{A})_{i,j}^2$$
$$g_y^2 = \sum_{i,j} (\mathbf{G}_y * \mathbf{A})_{i,j}^2$$
$$g_x g_y = \sum_{i,j} (\mathbf{G}_x * \mathbf{A})_{i,j} (\mathbf{G}_y * \mathbf{A})_{i,j}$$

and the sums are over each of the cells of the window. Then the eigenvalues λ_1 and λ_2 of Z and we pick $\lambda = \min(\lambda_1, \lambda_2)$. If this lambda is higher than some tolerance value, then we will assume that this window has a Shi-Tomasi feature. Otherwise we will assume that this window has spurious eigenvalues, resulting from numerical errors and should be ignored.

18.8 Particle Filter

The particle filter [4] is an algorithm used to estimate arbitrary probability distributions. Intutively, it is also very easy to understand. Assume an initial set of N evenly spaced particles, with each particle having a weight of $\frac{1}{N}$. Then perform a measurement y. For each particle, calculate the conditional probability $p(y \mid \phi_i)$, for $1 \leq i \leq N$, where ϕ_i is the position of the ith particle. Then set the new weight of the ith particle as $p(y \mid \phi_i)$, and normalize the weights of all particles. After a certain number of iterations, the value of the ith weight will start to resemble the probability of an observation being sampled with value ϕ_i. The set of particles will therefore start to look like the probability distribution from where the observations are sampled.

The particle filter is extremely useful when we are dealing with non-Gaussian distributions or when we want to test a set of different hypothesis. Unfortunately it requires knowing the conditional probability of observations to some degree of certainty, which will not always be feasible. Luckily, in our application this is not an issue, and therefore, we will ignore this problem for the remainder of the document.

18.9 Mean Shift algorithm

Assume we have two consecutive frames from the same camera. In the first frame we have an object and a bounding box around the object. In the next frame the object has moved by a small amount and we would like to find the bounding box that most closely matches the shape inside the original image. The mean shift algorithm [8] solves this problem by creating a "likelihood" image that relates both frames. The mean shift algorithm then takes an initial seed and moves it in the direction of where the "likelihood" image becomes dense, i.e. where similarity between the frames increases. After a number of iterations the seed is guaranteed to converge to a local maximum. This point will be the new center of the bounding box.

18.10 MonoSLAM

The necessary foundations for MonoSLAM were actually set up years prior to its introduction in the paper [9]. MonoSLAM was then properly introduced in [10]. Some work went into it during the next few years, with features like inverse depth parametrization [7] and scaleless runs being added to the algorithm.

MonoSLAM assumes a zero acceleration movement model. This doesn't mean that the algorithm assumes that the agent will never be submitted to acceleration. Instead it assumes that the model to generate background model states never changes the velocity of the agent. This assumption makes sense as for short enough intervals agents will rarely experience acceleration between frames. If a given agent has position vector r^W, orientation quaternion q^{WR}, velocity vector v^W and angular velocity vector ω^W, then the integration model, for timestep Δt is:

$$\hat{x} = \begin{pmatrix} r^W_{\text{new}} \\ q^{WR}_{\text{new}} \\ v^W_{\text{new}} \\ \omega^W_{\text{new}} \end{pmatrix} = \begin{pmatrix} r^W + v^W \Delta t \\ q^{WR} \times \mathbf{q}(\omega^W \Delta t) \\ v^W \\ \omega^W \end{pmatrix}$$

Our background model state and covariance are then:

$$x^b = \begin{pmatrix} \hat{x} \\ \hat{y}_1 \\ \hat{y}_2 \\ \vdots \end{pmatrix}$$

$$\mathbf{P}^b = \begin{pmatrix} \mathbf{P}_{x,x} & \mathbf{P}_{x,y_1} & \mathbf{P}_{x,y_2} & \cdots \\ \mathbf{P}_{y_1,x} & \mathbf{P}_{y_1,y_1} & \mathbf{P}_{y_1,y_2} & \cdots \\ \mathbf{P}_{y_2,x} & \mathbf{P}_{y_2,y_1} & \mathbf{P}_{y_2,y_2} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

where $\hat{y}_1$, $\hat{y}_2$, ... are the set of image features in 3D space. The set of observations is the result of applying the Mean Shift algorithm to find new 2D positions for each feature. The observation operator is then just the projection of the 3D image features into 2D space using the pinhole camera model (or wide angle, depending on the context). The observation covariance matrix is derived experimentally for the camera being used.

Then, a simplified description of the MonoSLAM algorithm is as follows:

- We wait for an image from the camera.
- As soon as we get an image we run it through the Sobel Operator.
- We then calculate the best Shi-Tomasi feature near the principal point of the camera.
- Using several consecutive images we estimate the position of this feature in the interval going from 0.5 meters to 5 meters using the particle filter. (This estimation can be aided by using previosuly known features with correct depth estimation during the filtering procedure).
- We then use the Mean Shift algorithm to estimate how much each of the features has moved in the 2D space.
- We use this information to feed an Extended Kalman Filter that updates the agent's knowledge of itself and its environment
- We enter into wait mode again.

18.11 Results

[10] presents a benchmark for MonoSLAM performance. The author ran the experiments in a 1.6 GHz Pentium M processor, much slower than today's CPUs. He also presented a breakdown of which parts of the algorithm caused which amount of slowdown. Here are the results:

- Image loading and administration: 2 ms
- Image correlation searches: 3 ms
- Kalman Filter update: 5 ms
- Feature initialization search: 4 ms
- Graphical rendering: 5 ms

for a grand total of 19 ms! This number is much lower than 33 ms, which is the slowest an algorithm that strives to run at 30 frames per second can go. This means that even if a frame takes somewhat longer to process, it is very unlikely that we will still get dropped frames.

References

[1] Computer science notes. `http://www.sci.utah.edu/~gerig/CS6640-F2012/Materials/pseudoinverse-cis61009sl10.pdf`. Accessed: 2018-11-18.

[2] Edge detection. `https://www.cs.auckland.ac.nz/courses/compsci373s1c/PatricesLectures/Edge%20detection-Sobel_2up.pdf`. Accessed: 2018-11-18.

[3] Roinets 2 - amplitude envelope connectivity analysis. `https://ohba-analysis.github.io/osl-docs/matlab/osl_example_roinets_2_manual.html`. Accessed: 2018-11-18.

[4] M.S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, 2002.

[5] R. N. Bannister. A review of operational methods of variational and ensemble-variational data assimilation. *Quarterly Journal of the Royal Meteorological Society*, 143(703):607–633, jan 2017.

[6] Angela Benedetti. Tangent linear and adjoint models for variational data assimilation. `https://software.ecmwf.int/wiki/download/attachments/31916718/Benedetti-Tangent_linear_adjoint_models.pdf?api=v2`, mar 2014. Accessed: 2018-11-18.

[7] J. Civera, A.J. Davison, and J. Montiel. Inverse depth parametrization for monocular SLAM. *IEEE Transactions on Robotics*, 24(5):932–945, oct 2008.

[8] D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, may 2002.

[9] Andrew J. Davison. Real-time simultaneous localisation and mapping with a single camera. In *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*, ICCV '03, pages 1403–, Washington, DC, USA, 2003. IEEE Computer Society.

[10] Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse. MonoSLAM: Real-time single camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1052–1067, jun 2007.

[11] Muhammad Mubashar Dogar, Fred Kucharski, and Syed Azharuddin. Study of the global and regional climatic impacts of ENSO magnitude using SPEEDY AGCM. *Journal of Earth System Science*, 126(2), mar 2017.

[12] J. Engel, J. Sturm, and D. Cremers. Semi-dense visual odometry for a monocular camera. In *IEEE International Conference on Computer Vision (ICCV)*, Sydney, Australia, December 2013.

[13] Ralf Giering and Thomas Kaminski. Recipes for adjoint code construction. *ACM Transactions on Mathematical Software*, 24(4):437–474, dec 1998.

[14] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2004.

[15] P. L. Houtekamer and Fuqing Zhang. Review of the ensemble kalman filter for atmospheric data assimilation. *Monthly Weather Review*, 144(12):4489–4532, dec 2016.

[16] Brian R. Hunt, Eric J. Kostelich, and Istvan Szunyogh. Efficient data assimilation for spatiotemporal chaos: A local ensemble transform kalman filter. *Physica D: Nonlinear Phenomena*, 230(1-2):112–126, jun 2007.

[17] Jan Mandel. A brief tutorial on the ensemble kalman filter, 2009.

[18] Takemasa Miyoshi. Research activities of the local ensemble transform kalman filter (letkf) at jma. `https://slideplayer.com/slide/12067418/`, 2007. Accessed: 2018-11-18.

[19] Franco Molteni and Fred Kucharski. Description of the ictp agcm (speedy) version 40. `http://users.ictp.it/~kucharsk/speedy_description/km_ver40_appendixA.pdf`. Accessed: 2018-11-18.

[20] Alexey Radul. Introduction to automatic differentiation. `https://alexey.radul.name/ideas/2013/introduction-to-automatic-differentiation/`, aug 2013. Accessed: 2018-11-18.

[21] Jianbo Shi and Tomasi. Good features to track. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition CVPR-94*. IEEE Comput. Soc. Press, 1994.

[22] R. Swarninathan and S.K. Nayar. Non-metric calibration of wide-angle lenses and polycameras. In *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*. IEEE Comput. Soc, 1999.

[23] Gabriel A. Terejanu. Extended kalman filter tutorial. `https://www.cse.sc.edu/~terejanu/files/tutorialEKF.pdf`. Accessed: 2018-11-18.

[24] Michael K. Tippett, Jeffrey L. Anderson, Craig H. Bishop, Thomas M. Hamill, and Jeffrey S. Whitaker. Notes and correspondence - ensemble square root filters. *Monthly Weather Review*, 131:1485–1490, july 2003. Accessed: 2018-11-18.

[25] Wei-Chuan Zhang and Peng-Lang Shui. Contour-based corner detection via angle difference of principal directions of anisotropic gaussian directional derivatives. *Pattern Recognition*, 48(9):2785–2797, sep 2015.