Parallel Filter Algorithms for Data Assimilation in Oceanography

von Lars Nerger

Dissertation

zur Erlangung des Grades
eines Doktors der Naturwissenschaften
— Dr. rer. nat. —

Angefertig am
Alfred-Wegener-Institut
für Polar und Meeresforschung
Bremerhaven

Vorgelegt im Fachbereich 3 (Mathematik & Informatik)
der Universität Bremen
im Dezember 2003

Datum des Promotionskolloquiums: 12. Februar 2004

Gutachter: Prof. Dr. Wolfgang Hiller (Universität Bremen und
Alfred-Wegener-Institut Bremerhaven)
Dr. Jens Schröter (Alfred-Wegener-Institut Bremerhaven)

Abstract

A consistent systematic comparison of filter algorithms based on the Kalman filter and intended for data assimilation with high-dimensional nonlinear numerical models is presented. Considered are the Ensemble Kalman Filter (EnKF), the Singular Evolutive Extended Kalman (SEEK) filter, and the Singular Evolutive Interpolated (SEIK) filter. Within the two parts of this thesis, the filter algorithms are compared with a focus on their mathematical properties as Error Subspace Kalman Filters (ESKF). Further, the filters are studied as parallel algorithms. This study includes the development of an efficient framework for parallel filtering.

In the first part, the filter algorithms are motivated in the context of statistical estimation. The unified interpretation of the algorithms as Error Subspace Kalman Filters provides the basis for the consistent comparison of the filter algorithms. The efficient implementation of the algorithms is discussed and their computational complexity is compared. Numerical data assimilation experiments with a test model based on the shallow water equations show how choices of the assimilation scheme and particular state ensembles for the initialization of the filters lead to significant variations of the data assimilation performance. The relation of the data assimilation performance to different qualities of the predicted error subspaces is demonstrated by a statistical examination of the predicted state covariance matrices. The comparison of the filters shows that problems of the analysis equations are apparent in the EnKF algorithm due to the Monte Carlo sampling of ensembles. In addition, the SEIK filter appears to be a numerically very efficient algorithm with high potential for use with nonlinear models.

The application of the EnKF, SEEK, and SEIK algorithms on parallel computers is studied in the second part. The parallelization possibilities of the different phases of the filter algorithms are examined. In addition, a framework for parallel filtering is developed which allows to combine filter algorithms with existing numerical models requiring only minimal changes to the source code of the model. The framework has been used to combine the parallel filter algorithms with the 3-dimensional finite element ocean model FEOM. Numerical data assimilation experiments are utilized to assess the parallel efficiency of the filtering framework and the parallel filters. The experiments yield an excellent parallel efficiency for the filtering framework. Furthermore, the framework and the filter algorithms are well suited for application to realistic large-scale data assimilation problems.

Contents

Introduction

Simulating the ocean general circulation provides the possibility to improve the understanding of climate relevant phenomena in the ocean. Absolute currents can be simulated which determine, for example, oceanic heat transports. Furthermore, the stability and variability of oceanic flows can be examined.

The numerical models used for simulating the ocean are based on physical first principles formulated by partial differential equations. Due to the discretization, models of high dimension arise. In addition, several different fields have to be modeled like, temperature, salinity, velocities, and the sea surface elevation. These large-scale ocean models are computationally demanding and hence require the use of parallel computers to cope with the huge memory and computing requirements. Despite their complexity, the models comprise several errors. Due to the finite resolution of the discretization, there are unresolved processes. These remain either unmodeled or are considered in parameterized form. Some processes are not included in the model physics or base on empirical formulas. The numerical solution itself will also cause errors. Apart from this, the model inputs also contain errors. That is, the model initialization is not exact and inputs during the simulation are uncertain, like fresh water inflows from rivers or interactions with the atmosphere, e.g. by the wind over the ocean.

A different source of information about the ocean is provided by observational data. Nowadays, there are many observations of the ocean provided by satellites like TOPEX/POSEIDON, or the more recent satellite missions Envisat and Jason-1. These satellites measure the sea surface height and temperature. Wind speeds and directions at the sea surface are measured by other satellites like QuikSCAT. In addition to satellite data, in situ measured observations are available. These include, e.g., temperatures and salinities at different depths, or current measurements from ships, moored instruments or drifting buoys. Despite the amount of available measurements, the observational data are sparse in space as well as in time. While there are many measurements at the ocean surface a relative small amount of information is provided about the interior of the ocean. Thus, the available observations do not suffice to provide a complete picture of the ocean.

To obtain an enhanced knowledge about the ocean, the information provided by numerical models and observational data should be used together. The combination of a numerical model with observations to determine the state of the modeled system is denoted inverse modeling. In meteorology and oceanography, the quantitative framework to solve inverse problems is known as “data assimilation”. This technique incorporates – assimilates – observational data into a numerical model to improve the ocean state simulated by the model.

There are currently two main approaches to data assimilation which are either based on optimal control theory or on estimation theory, see e.g. [77, 24]:

- *Variational data assimilation* – This technique uses a criterion measuring the misfit between model and observations. This criterion, typically denoted the cost function, has to be minimized by adjusting so called control variables of the model. These are usually initial conditions or certain parameters of the model such as the wind stress or heat flux. Variational data assimilation is based on the theory of optimal control. The most common method is the so called adjoint method, see [14, 78], which is widely used in oceanography, see e.g. [93, 76]. A related variational method is the representer method [3, 10].

- *Sequential data assimilation* – This technique is based on estimation theory and represents a filter method. The observations and the model prediction of the state are combined using weights computed from the estimated uncertainties of both the predicted model state and the observational data. The schemes used for sequential data assimilation are mostly based on the Kalman filter [41, 42]. An alternative approach is represented by particle filters, see [2, 55, 85, 47].

The advantage of sequential data assimilation algorithms is their flexibility. While the adjoint method requires to integrate the numerical model and its adjoint multiple times over the time interval of interest, the sequential schemes assimilate observational data at the time instance at which the data becomes available. Thus, with sequential algorithms it is not required to restart the assimilation cycle when new observations are provided. In addition, an adjoint of the numerical model is not required by the sequential methods. Also the potential for parallelization is higher for the algorithms based on the Kalman filter.

The first approaches to apply the Kalman filter in oceanography relate back to the middle of the 1980's. The Kalman filter is only suited for linear systems and the application of the full Kalman filter is not feasible for realistic large-scale numerical ocean models. During the last decade several algorithms have been developed on the basis of the Kalman filter which reduce the computational requirements of the Kalman filter to feasible limits and promise to handle nonlinearity in a better way.

One of the newly developed algorithms is the Ensemble Kalman Filter (EnKF), introduced by Evensen [17]. This filter is based on a Monte Carlo approach and, due to its apparent simplicity, already widely used in oceanography and meteorology (see, e.g. [18] for a review of applications of the EnKF). In addition, some variants of the EnKF have been proposed [34, 1, 5, 94]. Alternative algorithms are the SEEK and SEIK filters, introduced by Pham [65, 68]. These filters represent the estimated error statistics by a low-rank matrix. Some variants of these filters have been proposed which permit to further reduce the computational requirements [32, 33]. The SEEK filter has been applied in several studies, e.g. [90, 9, 63, 7, 6], and some applications of the SEIK algorithm have been reported [66, 33, 83]. Other approaches to a simplified filter are the reduced-rank square root Kalman (RRSQRT) filter by Verlaan and Heemink [88] and the concept of error subspace statistical estimation introduced by Lermusiaux and Robinson [49, 50].

The computational requirements of data assimilation problems is generally much higher than for numerical ocean models alone. Thus, the use of parallel computers is strongly required when data assimilation is performed with realistic large-scale numerical models. The algorithms based on the Kalman filters offer a high potential for parallelization. The application of the filter algorithms on parallel computers has been discussed for the Ensemble Kalman filter by Keppenne and Rienecker [44, 45] and by Houtekamer and Mitchell [36]. Some approaches have also been investigated in the context of the RRSQRT algorithm [73, 70].

Besides the use of parallel computers there is the requirement to combine data assimilation algorithms with existing models to obtain a data assimilation system. This should be possible with minimal changes to the model source code. Verlaan [87] discussed an abstract coupling between a model and filter algorithm. In addition, the programs SESAM [75] and PALM [60] provide interface structures based on strongly different concepts.

In this work a consistent systematic comparison of filter algorithms based on the Kalman filter is presented. Considered are the Ensemble Kalman filter and the SEEK filter. The former algorithm represents the Monte Carlo approach to filtering while the latter algorithm uses a low-rank approximation to represent the error statistics of the model. Further, the SEIK filter, which unites aspects of both approaches, is included in the study. Besides the comparison, parallel variants of the algorithms are developed and discussed. In addition, an efficient framework for parallel filtering is introduced. The framework defines an application program interface to combine the filter algorithms with existing numerical models. To test the efficiency of the framework, it is used to combine the filter algorithms with the three-dimensional finite element ocean model FEOM which has been recently developed at the Alfred Wegener Institute [12].

The new unified interpretation of the filter algorithms as Error Subspace Kalman Filters (ESKF) provides the basis to compare the algorithms consistently. The interpretation corresponds to the concept of error subspace statistical estimation [49]. The experimental study of the ESKF algorithms under identical conditions presents the first quantitative comparison of these algorithms. It also shows the influence of higher order sampling schemes. Heemink et al. [31] performed a numerical comparison of the RRSQRT and EnKF algorithms using a 2-dimensional advection-diffusion equation. In addition, the EnKF algorithm was compared with the SEEK filter [7] using a model of the North Atlantic. In this study, however, the experimental configurations differed for the two algorithms rendering the results difficult to interpret.

The parallelization of the SEEK and SEIK filters has not yet been discussed. Furthermore, a separated parallelization of the filter algorithms and parallel model tasks is hardly considered [70, 60]. The filtering framework presented in this work is, on the one hand, simpler than the existing PALM coupler interface [60], on the other hand it is more efficient than SESAM [75]. The application of filter algorithms to a three-dimensional finite element ocean model has not yet been reported. The studies presented in this work, which use an idealized configuration of FEOM, yield promising results proving feasibility of the algorithms also for realistic model configurations.

Outline

This work is subdivided into two parts. The first considers filter algorithms based on the Kalman filter as sequential algorithms with a focus on their mathematical properties. The second part discusses the filters as parallel algorithms.

In part I, the fundamentals of data assimilation are introduced in chapter 1. In chapter 2, the filter algorithms based on the Kalman filter and intended for application to large-scale nonlinear numerical models are motivated, presented, and discussed as Error Subspace Kalman Filters (ESKF) in the context of statistical estimation. Subsequently, in chapter 3, the ESKF algorithms are compared under the aspect of their application to large-scale nonlinear models. The efficient implementation and the numerical complexity of the algorithms are also discussed in this chapter. To assess the capabilities of the ESKF algorithms experimentally, the filters are applied in identical twin experiments to an oceanographic test model in chapter 4. Part I is concluded by chapter 5 summarizing the findings of the study of Error Subspace Kalman Filters.

Part II is commenced in chapter 6 with an overview and motivation of the application of ESKF algorithms as parallel algorithms. The parallelization possibilities of the ESKF algorithms are examined in chapter 7. Here different approaches are discussed and resulting parallel algorithms are presented. Chapter 8 introduces a framework for parallel filtering. This framework defines an application program interface which permits to combine the parallel filter algorithms with existing numerical models requiring minimal changes to the model source code. In Chapter 9 the parallel efficiency of the filtering framework and the parallel filter algorithms is studied. For this, the framework is used to combine the filter algorithms with the finite element model FEOM. Twin experiments are performed to assess the parallel efficiency of both the framework and the algorithms. Further, the data assimilation capabilities of the ESKF algorithms when applied to a three-dimensional model are examined. The results of this part are summarized and conclusions are drawn in Chapter 10 which completes part II.

Part I

Error Subspace Kalman Filters

Chapter 1

Data Assimilation

1.1 Overview

Data assimilation is the framework to combine the information provided by measurements with a numerical model describing the physical processes of the considered geophysical system. There are three different application types of data assimilation. First, the future state of the physical system can be computed based on observations available until the present time. This application type is denoted as forecasting. Second, the current state can be estimated on the basis of all observations available until now. This situation is referred to as filtering or now-casting. The third application type is smoothing or re-analysis. Here the state trajectory in the past is estimated based on all observations available until the present time.

The technique of data assimilation originated in meteorology from the need to provide accurate weather forecasts. From the first steps of objective analysis of observational data about 50 years ago, the techniques evolved toward the current assimilation methods. A review on this history is given by Ghil and Malanotte-Rizzoli [25]. The method of optimal interpolation (see e.g. [51]), which was the most widely used method for operational numerical weather prediction in 1991 when this article was published, is today replaced by 4D-Var, see e.g. [69]. This is the space and time dependent variational data assimilation using the adjoint method. In addition, approaches to the application of sequential algorithms based on the Kalman filter exist [20, 21].

The situation for data assimilation in physical oceanography is different from that in meteorology. The spatial scales in the ocean are smaller than in the atmosphere. In contrast to this, the time scales are larger. In addition, the amount of observational data of the ocean is significantly smaller than the quantity of atmospheric measurements. Due to this, oceanographic data assimilation is a rather young discipline motivated by the improvement in the understanding of the dynamics of ocean circulation. However, the availability of remotely sensed observations from satellites increased the amount of data significantly motivating further the application of data assimilation in oceanography (see e.g. [16] for a review on several data assimilation methods used with ocean models). Today, there are first attempts for operational oceanography or ocean forecasting which involve advanced data assimilation algorithms, e.g. by the projects DIADEM [13] and MERCATOR [54].

Data assimilation algorithms are currently characterized by two main approaches. The first is variational data assimilation which is based on optimal control theory. One representative of this approach is the widely used adjoint method. Because of its current importance, this technique will be reviewed in the following section. The second approach is provided by sequential data assimilation algorithms. These filter methods are based on estimation theory and are typically derived from the Kalman filter [41, 42]. These algorithms are the subject of this work. Section 1.3 provides an overview on the sequential data assimilation algorithms based on the Kalman filter. The mathematical foundations of these algorithms are introduced in Chapter 2.

1.2 The Adjoint Method

The adjoint method is a variational technique aiming at the minimization of an empiric criterion measuring the misfit between a model and the observations. It is typically employed as a smoothing method or to provide a state estimate used to compute a forecast. The adjoint method is derived here according to the derivation by Le Dimet and Talagrand [14]. The notations follow the unified notation proposed by Ide et al. [37].

The principle of the adjoint method is as follows:
We consider a physical system which is represented by the state vector $\mathbf{x}(t) \in \mathcal{S}$ where $\mathcal{S}$ is a Hilbert space with inner product $< , >$. The time evolution of the state is described by the model

$$\frac{d\mathbf{x}(t)}{dt} = M[\mathbf{x}(t)] \tag{1.1}$$

with the initial condition

$$\mathbf{x}(t_0) = \mathbf{x}_0 \ . \tag{1.2}$$

In addition, observations $\{\mathbf{y}^o(t_i)\}$ of the state will be available at some time instances $\{t_i,\ i = 1, \ldots, k\}$.

Let the misfit between the state and the observations be described by the scalar cost functional J given by

$$J[\mathbf{u}] = \frac{1}{2} \sum_{i=1}^{k} < \mathbf{y}^o(t_i) - \mathbf{x}(t_i),\ \mathbf{y}^o(t_i) - \mathbf{x}(t_i) > \tag{1.3}$$

where $\mathbf{u}$ is the vector of control variables. For simplicity we consider the case that the initial state is used as the control variables:

$$\mathbf{u} = \mathbf{x}(t_0) \tag{1.4}$$

The problem of variational data assimilation is now: Find the optimal vector $\tilde{\mathbf{u}}$ of control variables which minimizes the cost functional J:

$$J[\tilde{\mathbf{u}}] = \min_{\mathbf{u}} J[\mathbf{u}] \tag{1.5}$$

To minimize J with respect to $\mathbf{u}$, e.g. by the quasi-Newton optimization method, the gradient $\nabla_{\mathbf{u}} J$ has to be computed. The gradient is defined by

$$\delta_{\mathbf{u}} J =< \nabla_{\mathbf{u}} J,\ \delta\mathbf{u} > \ . \tag{1.6}$$

where $\delta_{\mathbf{u}} J$ is the first order variation of J with respect to $\mathbf{u}$. $\delta\mathbf{u}$ is the perturbation of $\mathbf{u}$. From equation (1.3) the first order variation of J resulting from a perturbation $\delta\mathbf{x}(t_0)$ is given by

$$\delta_{\mathbf{u}} J = \sum_{i=1}^{k} < \mathbf{y}^o(t_i) - \mathbf{x}(t_i),\ \delta\mathbf{x}(t_i) > \tag{1.7}$$

where the first order variations $\{\delta\mathbf{x}(t_i)\}$ are related to the perturbation $\delta\mathbf{x}(t_0)$ by

$$\delta\mathbf{x}(t_i) = R(t_i, t_0)\delta\mathbf{x}(t_0),\ i = 1, \ldots, k \ . \tag{1.8}$$

$R(t_i, t_0)$ is the resolvent of the linearization

$$\frac{d}{dt}\delta\mathbf{x}(t) = \mathbf{M}(t)\delta\mathbf{x}(t) \tag{1.9}$$

of equation (1.1) about the state $\mathbf{x}(t)$. Here $\mathbf{M}(t)$ is the linearized model operator. Equation (1.9) is also denoted the *tangent linear model*. The resolvent $R(t_i, t_0)$ is the linear operator obtained by integrating equation (1.9) from time t_0 to time t_i under the initial condition $\delta\mathbf{x}(t_0) = \delta\mathbf{u}$.

For any continuous linear operator L on $\mathcal{S}$ exists a linear operator $L^\dagger$ on $\mathcal{S}$ defined by

$$< \mathbf{a},\ L\mathbf{b} >=< L^\dagger\mathbf{a},\ \mathbf{b} >,\ \forall\mathbf{a}, \mathbf{b} \in \mathcal{S} \ . \tag{1.10}$$

$L^\dagger$ is denoted the *adjoint operator* of L. Introducing the adjoint resolvent $R^\dagger(t_i, t_0)$, equation (1.7) can be written as

$$\delta_{\mathbf{u}} J = \sum_{i=1}^{k} < R^\dagger(t_i, t_0)\left[\mathbf{y}^o(t_i) - \mathbf{x}(t_i)\right],\ \delta\mathbf{x}(t_i) > \ . \tag{1.11}$$

Hence, the gradient of J with respect to $\mathbf{u}$ is, according to equations (1.6) and (1.4),

$$\nabla_{\mathbf{u}} J = \sum_{i=1}^{k} R^\dagger(t_i, t_0)\left[\mathbf{y}^o(t_i) - \mathbf{x}(t_i)\right] \tag{1.12}$$

The adjoint resolvent can be determined in the following way: The *adjoint model* to the tangent linear model (1.9) is given by

$$\frac{d}{dt}\delta\mathbf{x}^\dagger(t) = -\mathbf{M}^\dagger(t)\delta\mathbf{x}^\dagger(t) \tag{1.13}$$

where $\delta\mathbf{x}^\dagger(t) \in \mathcal{S}$ and $\mathbf{M}^\dagger(t)$ is the adjoint of $\mathbf{M}(t)$. Now it can be shown, see [14], that the resolvent $S(t_0, t_i)$ of equation (1.13) is given by the adjoint resolvent of equation (1.8):

$$S(t_0, t_i) = R^\dagger(t_i, t_0) \tag{1.14}$$

Thus, the gradient of J is finally obtained as

$$\nabla_{\mathbf{u}} J = \sum_{i=1}^{k} S(t_0, t_i) \left[\mathbf{y}^o(t_i) - \mathbf{x}(t_i)\right] \tag{1.15}$$

The term $S(t_0, t_i) \left[\mathbf{y}^o(t_i) - \mathbf{x}(t_i)\right]$ is evaluated by integrating the adjoint model (1.13) backward in time from t_i to t_0 with the initial condition $\delta\mathbf{x}^\dagger(t_i) = \mathbf{y}^o(t_i) - \mathbf{x}(t_i)$. Since equation (1.13) is linear, a single backward integration suffices to compute the the gradient $\nabla_{\mathbf{u}} J$. For this the integration is started at time t_k with the initial condition $\delta\mathbf{x}^\dagger(t_k) = \mathbf{y}^o(t_k) - \mathbf{x}(t_k)$. During the backward integration the term $\mathbf{y}^o(t_i) - \mathbf{x}(t_i)$ is added to the current value $\delta\mathbf{x}^\dagger(t_i)$ at time instants t_i where observations are available.

Summarizing, the adjoint method to compute the optimal initial conditions is given by the iterative algorithm:

1. Choose some estimate $\mathbf{x}_0$ of the initial state vector: $\mathbf{x}(t_0) = \mathbf{x}_0$.
2. For $j = 1, \ldots$ loop:
3. Integrate the model (1.1) from t_0 to t_k. Store the obtained state trajectory.
4. Evaluate the cost functional J according to equation (1.3).
5. Integrate the adjoint model (1.13) backward in time from t_k to t_0 starting from $\delta\mathbf{x}^\dagger(t_k) = \mathbf{y}^o(t_k) - \mathbf{x}(t_k)$. Add $\mathbf{y}^o(t_i) - \mathbf{x}(t_i)$ to $\delta\mathbf{x}^\dagger(t_i)$ at each observation time. Then, according to equation (1.15), it is $\nabla_{\mathbf{u}} J = \delta\mathbf{x}^\dagger(t_o)$.
6. If $\nabla_{\mathbf{u}} J \leq \epsilon$ for some condition ϵ, exit the loop over j.
7. Update the initial condition according to the chosen optimization algorithm, e.g. quasi-Newton.
8. End of the loop over j.

Remarks on the adjoint method:
Remark 1: The formulation of the adjoint method can be extended to optimize, e.g., physical parameters or lateral boundary conditions. In addition, the method can be extended to handle observations which are functions of the state vector. Thus, it is not required that the complete state vector itself is observed.
Remark 2: To apply the adjoint method, the adjoint operator $\mathbf{M}^\dagger(t)$ has to be implemented. For large-scale nonlinear models the propagation operator M is implicitly defined by its implementation in the source code of the model. Hence, also the adjoint operator has to be implemented as an operator rather than as an explicit matrix. The implementation is a difficult task. It can, however, be simplified by automatic differentiation tools like TAMC, see [53].
Remark 3: The adjoint method does not provide an estimate of the error of the obtained optimal control variables. To obtain an error estimate, the Hessian matrix of the cost function J has to be determined [95].

Remark 4: The adjoint method requires to integrate the model and the adjoint model multiple times during the optimization process. These integrations are the most time consuming part of the algorithm.
Remark 5: To evaluate the adjoint model operator $\mathbf{M}^{\dagger}(t)$, the state trajectory of the forward integration (point 2) has to be stored. If the time integration is performed over long time intervals with large-scale models, huge memory requirements will result.

1.3 Sequential Data Assimilation

Sequential data assimilation algorithms combine the predicted state estimate of a model with observations at the time when the observational data become available. The combination, denoted analysis, is computed using weights obtained from the estimated errors of both the model state and the observations. The computed state estimate can be used to perform a model forecast. Also it is possible to formulate smoothing algorithms which also modify the model state in the past on the basis of a newly available observation, see [86]. This work will focus on filtering, that is, the current state is estimated using only the observations available up to the present time.

Over the recent years there has been an extensive development of filter algorithms based on the Kalman filter (KF) [41, 42] in the atmospheric and oceanic context. These filter algorithms are of special interest due to their simplicity of implementation, e.g. no adjoint operators are required, and their potential for efficient use on parallel computers with large-scale geophysical models [45]. In addition, an error estimate is provided by the filter algorithms in form of an estimated error covariance matrix of the model state.

The classical KF and the extended Kalman filter (EKF), see [38], share the problem that for large-scale models the requirements of computation time and storage are prohibitive. This is due to the explicit treatment of the error covariance matrix of the model state. Furthermore, the EKF shows deficiencies with the nonlinearities appearing, e.g., in oceanographic systems [15]. Due to this, algorithms are required which reduce the memory and computation requirements and provide better abilities to handle nonlinearity.

There have been different working directions over the recent years. One approach is based on a low-rank approximation of the state error covariance matrix of the EKF in order to reduce the computational costs. Using gradient approximations of the linearized model which is required to evolve the covariance matrix, these algorithms also show better abilities to handle nonlinearity than the EKF. Examples of low-rank filters are the Reduced Rank Square-Root (RRSQRT) algorithm [88] and the Singular Evolutive Extended Kalman (SEEK) filter [68]. An alternative approach is to employ an ensemble of model states to represent the error statistics which are treated in the EKF by the state estimate and its covariance matrix. An example is the Ensemble Kalman filter (EnKF) [17, 8] which applies a Monte Carlo method to forecast the error statistics. For an improved treatment of nonlinearities, Pham et al. [67] introduced the Singular Evolutive Interpolated (SEIK) filter as a variant of the SEEK filter. It combines

the low-rank approximation with an ensemble representation of the covariance matrix. This idea has also been followed in the concept of Error Subspace Statistical Estimation (ESSE) [49].

The major part of the computation time in data assimilation with filter algorithms is spent for the prediction of error statistics using the linearized or the nonlinear model. Thus, the efficiency of a filter algorithm will be determined by its ability to yield sufficiently good estimates with as few model evaluations as possible. In general, using a larger rank for the approximation of the state covariance matrix or a larger ensemble for its representation will provide a more reliable state estimate. In practice, the rank or ensemble size will be, however, limited by the available computing resources.

Chapter 2

Filter Algorithms

2.1 Introduction

This chapter introduces the mathematical foundations of filter algorithms based on the Kalman filter. In addition, the equations of several approximating algorithms are motivated and related to the extended Kalman filter. The focus lies on the Ensemble Kalman Filter [17], the Singular Evolutive Extended Kalman (SEEK) filter [68] and the Singular Evolutive Interpolated Kalman (SEIK) filter [67][1]. The EnKF and SEEK filters are representative for the two approaches of low-rank and ensemble filters. The SEIK filter is considered because it unites aspects of both approaches. The relation of these filters to other approximating filter algorithms will be discussed. The SEEK, EnKF, and SEIK algorithms approximate the full error space of the EKF by an error subspace. In addition, all algorithms apply the analysis equations of the Kalman filter. For this reason, it will be referred to the algorithms as Error Subspace Kalman Filters (ESKF).

The filter algorithms are presented and discussed based on the probabilistic viewpoint similar to Cohn [11] but with a focus on nonlinear large-scale systems. For ease of comparison, the notations follow, as far as possible, the unified notation proposed by Ide et al. [37]. Section 2.2 introduces to the estimation theory. The Kalman filter and the extended Kalman filter are motivated and discussed in section 2.3. Subsequently, in section 2.4 the error subspace Kalman filter algorithms SEEK, EnKF, and SEIK are introduced and discussed. The discussion of the extended Kalman filter and the ESKF filters is performed assuming a linear relation between model fields and observations. The situation of nonlinearly related model fields and observations is discussed in section 2.5.

[1] The names of the latter two algorithms have a French origin with "evolutive" coming from the French word "évolutif" meaning evolving.

2.2 Statistical Estimation

We consider a physical system which is represented by its state $\mathbf{x}(t) \in \mathcal{S}$ where $\mathcal{S}$ is a Hilbert space. The state is described by a discrete numerical model governing the propagation of the discretized state $\mathbf{x}^t \in \mathcal{S}^n$, denoted the true state. Since the discrete model only approximates the true physics of the system, $\mathbf{x}^t$ is a random vector whose time propagation is given by the stochastic-dynamic time discretized model equation

$$\mathbf{x}_i^t = M_{i,i-1}[\mathbf{x}_{i-1}^t] + \boldsymbol{\eta}_i \ . \tag{2.1}$$

Here $M_{i,i-1}$ is a, possibly nonlinear, operator describing the state propagation between the two consecutive time steps $i-1$ and i. The vector $\boldsymbol{\eta}_i$ is the model error, which is assumed to be a stochastic perturbation with zero mean and covariance matrix $\mathbf{Q}_i$.

At discrete times $\{t_k\}$, each Δk time steps, observations are available as a vector $\mathbf{y}_k^o$ of dimension m_k. The true state $\mathbf{x}_k^t$ at time t_k is assumed to be related to the observation vector by the measurement model

$$\mathbf{y}_k^o = H_k[\mathbf{x}_k^t] + \boldsymbol{\epsilon}_k \ . \tag{2.2}$$

Here H_k is the forward measurement operator. It describes diagnostic variables, i.e., the observations which would be measured given the state $\mathbf{x}_k^t$. The vector $\boldsymbol{\epsilon}_k$ is the observation error consisting of the measurement error due to imperfect measurements and the representation error caused, e.g., by the discretization of the dynamics. $\boldsymbol{\epsilon}_k$ is a random vector. It is assumed to be of zero mean and covariance matrix $\mathbf{R}_k$ and uncorrelated with the model error $\boldsymbol{\eta}_k$.

The state sequence $\{\mathbf{x}_i^t\}$, prescribed by equation (2.1), is a stochastic process which is completely described by a probability density function $p(\mathbf{x}_i^t)$. The state sequence is a Markov process under the assumptions that the model error $\boldsymbol{\eta}_i$ is Gaussian and white in time $\{\mathbf{x}_i^t\}$. In this case, the time evolution of $p(\mathbf{x}_i^t)$ is described by the Fokker-Planck or forward Kolmogorov equation (see Jazwinski [38]), in time discretized form

$$p(\mathbf{x}_i) = p(\mathbf{x}_{i-1}) - \sum_{\alpha=1}^{n} \frac{\partial \left(p(\mathbf{x}_{i-1}) M_{i,i-1(\alpha)}(\mathbf{x}_{i-1}) \right)}{\partial \mathbf{x}_{i-1(\alpha)}} + \frac{1}{2} \sum_{\alpha,\beta=1}^{n} \frac{\partial^2 \left(p(\mathbf{x}_{i-1}) \mathbf{Q}_{(\alpha\beta)} \right)}{\partial \mathbf{x}_{i-1(\alpha)} \partial \mathbf{x}_{i-1(\beta)}} \tag{2.3}$$

where the Greek indices denote the components. In practice, the high dimensionality of realistic geophysical models prohibits the explicit solution of the Fokker-Planck-Kolmogorov equation. Nonetheless, it is possible to derive equations for statistical moments of the probability density like the mean and the covariance matrix, see, for example Jazwinski [38].

In general, the filtering problem is solved by the conditional probability density function $p(\mathbf{x}_k^t | \mathbf{Y}_k^o)$ of the true state given the observations $\mathbf{Y}_k^o = \{\mathbf{y}_0^o, \ldots, \mathbf{y}_k^o\}$ up to time t_k. In practice, it is not feasible to compute this density explicitly for large-scale models. Thus, one has to rely on the calculation of some statistics of the density. In the context of filtering usually the conditional mean is computed, which is also the minimum variance estimate, see Jazwinski [38].

In the following we will concentrate on sequential filter algorithms. That is, the algorithms consist of two phases: In the *forecast phase* the conditional probability density $p(\mathbf{x}^t_{k-\Delta k}|\mathbf{Y}^o_{k-\Delta k})$, or statistical moments of it, is evolved up to the time t_k when observations are available, yielding $p(\mathbf{x}^t_k|\mathbf{Y}^o_{k-\Delta k})$. Then, in the *analysis phase*, the conditional density $p(\mathbf{x}^t_k|\mathbf{Y}^o_k)$ is computed from the forecasted density and the observation vector $\mathbf{y}^0_k$. Subsequently the cycle of forecasts and analyses is repeated.

To initialize the filter sequence an initial density $p(\mathbf{x}^t_0|\mathbf{Y}^o_0)$ is required. In practice this density is unknown and a density estimate $p(\mathbf{x}_0)$ is used for the initialization.

2.3 The Extended Kalman Filter

For linear dynamic and measurement models, the Kalman filter (KF) [41, 42] is the minimum variance estimator if the initial probability density $p(\mathbf{x}^t_0)$ and the model error and observation error processes are Gaussian. To clarify the assumptions about the statistics of the model error, the observation error and the probability density of the model state, we will motivate the KF based on statistical estimation. With this we will also show the approximations which are required for the derivation of the Extended Kalman Filter. A detailed derivation of the KF in the context of statistical estimation is presented by Cohn [11] and several approaches toward the KF are discussed in Jazwinski [38].

First, let us consider linear dynamic and measurement operators. Thus, equations (2.1) and (2.2) can be written in matrix-vector form as

$$\mathbf{x}^t_k = \mathbf{M}_{k,k-\Delta k}\mathbf{x}^t_{k-\Delta k} + \boldsymbol{\eta}_k \ , \tag{2.4}$$

$$\mathbf{y}^o_k = \mathbf{H}_k\mathbf{x}^t_k + \boldsymbol{\epsilon}_k \ . \tag{2.5}$$

Here the linear operator $\mathbf{M}_{k,k-\Delta k}$ propagates the state vector from time step $k-\Delta k$ to time step k. We assume that the stochastic processes $\boldsymbol{\eta}_k$ and $\boldsymbol{\epsilon}_k$ are temporal white Gaussian processes with zero mean and respective covariance matrices $\mathbf{Q}_k$ and $\mathbf{R}_k$. Additionally, the probability density function $p(\mathbf{x}^t_k)$ is assumed to be Gaussian with covariance matrix $\mathbf{P}_k$, and all three processes are mutually uncorrelated. Denoting the expectation operator by $< >$, the assumptions are summarized as

$$\boldsymbol{\eta}_i \propto \mathcal{N}(\mathbf{0}, \mathbf{Q}_i) \ ; \quad < \boldsymbol{\eta}_i\boldsymbol{\eta}^T_j >= \mathbf{Q}_i\delta_{ij} \tag{2.6}$$

$$\boldsymbol{\epsilon}_k \propto \mathcal{N}(\mathbf{0}, \mathbf{R}_k) \ ; \quad < \boldsymbol{\epsilon}_k\boldsymbol{\epsilon}^T_l >= \mathbf{R}_k\delta_{kl} \tag{2.7}$$

$$\mathbf{x}^t_i \propto \mathcal{N}(\bar{\mathbf{x}}^t_i, \mathbf{P}_i) \ ; \tag{2.8}$$

$$< \boldsymbol{\eta}_k\boldsymbol{\epsilon}^T_k >= 0 \ ; \quad < \boldsymbol{\eta}_i(\mathbf{x}^t_i)^T >= 0 \ ; \quad < \boldsymbol{\epsilon}_k(\mathbf{x}^t_k)^T >= 0 \ , \tag{2.9}$$

where $\mathcal{N}(\mathbf{a}, \mathbf{B})$ denotes the normal distribution with mean $\mathbf{a}$ and covariance matrix $\mathbf{B}$ and δ_{kl} is the Kronecker delta with $\delta_{kl} = 1$ for $k = l$ and $\delta_{kl} = 0$ for $k \neq l$. Under assumptions (2.6) - (2.8) the corresponding probability densities are fully described by their two lowest statistical moments: the mean and the covariance matrix. Applying this property, the KF formulates the filter problem in terms of the conditional means and covariance matrices of the forecasted and analyzed state probability densities.

To derive the forecast equations for the KF only a part of assumptions (2.6) to (2.9) is required. Suppose the conditional density $p(\mathbf{x}^t_{k-\Delta k}|\mathbf{Y}^o_{k-\Delta k})$ at time $t_{k-\Delta k}$ is given in terms of the conditional mean

$$\mathbf{x}^a_{k-\Delta k} :=< \mathbf{x}^t_{k-\Delta k}|\mathbf{Y}^o_{k-\Delta k} > \ , \tag{2.10}$$

denoted *analysis state*, and the *analysis covariance matrix*

$$\mathbf{P}^a_{k-\Delta k} :=< (\mathbf{x}^t_{k-\Delta k} - \mathbf{x}^a_{k-\Delta k})(\mathbf{x}^t_{k-\Delta k} - \mathbf{x}^a_{k-\Delta k})^T|\mathbf{Y}^o_{k-\Delta k} > \ . \tag{2.11}$$

In the forecast phase, the KF evolves the density forward until time t_k. That is, the mean and covariance matrix of the probability density $p(\mathbf{x}^t_k|\mathbf{Y}^o_{k-\Delta k})$ are computed. The *forecast state* is the conditional mean $\mathbf{x}^f_k :=< \mathbf{x}^t_k|\mathbf{Y}^o_{k-\Delta k} >$. With the dynamic model equation (2.4) and the assumption that the model error has zero mean this leads to

$$\mathbf{x}^f_k = \mathbf{M}_{k,k-\Delta k}\mathbf{x}^a_{k-\Delta k} \ . \tag{2.12}$$

The expression for the corresponding *forecast covariance matrix* follows from equations (2.4), (2.12), and the assumption (2.9) that $\mathbf{x}^t_k$ and $\boldsymbol{\eta}_k$ are uncorrelated, as

$$\begin{aligned} \mathbf{P}^f_k &:= < (\mathbf{x}^t_k - \mathbf{x}^f_k)(\mathbf{x}^t_k - \mathbf{x}^f_k)^T|\mathbf{Y}^o_{k-\Delta k} > \\ &= \mathbf{M}_{k,k-\Delta k}\mathbf{P}^a_{k-\Delta k}\mathbf{M}^T_{k,k-\Delta k} + \mathbf{Q}_k \ . \end{aligned} \tag{2.13}$$

Equations (2.12) and (2.13) represent the forecast phase of the KF. Besides the assumption of uncorrelated processes $\mathbf{x}^t_k$ and $\boldsymbol{\eta}_k$ and unbiased model error no further statistical assumptions are required for the derivation of these equations, in particular the densities are not required to be Gaussian.

Suppose a vector of observations $\mathbf{y}^o_k \in \mathbb{R}^{m_k}$ to be available at time t_k. Then the analysis phase of the KF computes the mean and covariance matrix of the conditional density $p(\mathbf{x}^t_k|\mathbf{Y}^o_k)$ given the density $p(\mathbf{x}^t_k|\mathbf{Y}^o_{k-\Delta k})$ and the observation vector $\mathbf{y}^o_k$. Under the assumption that the error process $\boldsymbol{\epsilon}_k$ is white in time, the solution is given by Bayes' theorem as

$$p(\mathbf{x}^t_k|\mathbf{Y}^o_k) = \frac{p(\mathbf{y}^o_k|\mathbf{x}^t_k)p(\mathbf{x}^t_k|\mathbf{Y}^o_{k-\Delta k})}{p(\mathbf{y}^o_k|\mathbf{Y}^o_{k-\Delta k})} \ . \tag{2.14}$$

Since this relation only implies the whiteness of $\boldsymbol{\epsilon}_k$ it is also valid for nonlinear dynamic and measurement operators. Assumptions (2.6) to (2.9) are however required to derive the analysis equations as the mean and covariance matrix of the analysis density $p(\mathbf{x}^t_k|\mathbf{Y}^o_k)$. A lengthly calculation leads to the analysis state $\mathbf{x}^a_k$ and analysis covariance matrix $\mathbf{P}^a_k$ as

$$\mathbf{x}^a_k = \mathbf{x}^f_k + \mathbf{K}_k(\mathbf{y}^o_k - \mathbf{H}_k\mathbf{x}^f_k) \ , \tag{2.15}$$

$$\mathbf{P}^a_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}^f_k(\mathbf{I} - \mathbf{K}^T_k\mathbf{H}^T_k) + \mathbf{K}_k\mathbf{R}_k\mathbf{K}^T_k \tag{2.16}$$

$$= (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}^f_k \tag{2.17}$$

where $\mathbf{K}_k$ is denoted the *Kalman gain*. Equation (2.17) is only valid for a $\mathbf{K}_k$ given by

$$\mathbf{K}_k = \mathbf{P}^f_k\mathbf{H}^T_k(\mathbf{H}_k\mathbf{P}^f_k\mathbf{H}^T_k + \mathbf{R}_k)^{-1} \tag{2.18}$$

or, alternatively, if $\mathbf{R}_k$ is invertible,

$$\mathbf{K}_k = \mathbf{P}_k^a \mathbf{H}_k^T \mathbf{R}_k^{-1} \ . \tag{2.19}$$

Equations (2.15) to (2.18) complete the KF theory.

The Extended Kalman filter (EKF) is a first-order extension of the KF to nonlinear models as given by equations (2.1) and (2.2). Again it is based on the first two statistical moments of the probability density and on the probabilistic assumptions (2.6)-(2.9). The EKF equations are obtained by linearizing the dynamic and measurement operators around the most recent state estimate. We will consider here only the case of linear measurement operators. The use of nonlinear measurement operators is discussed in section 2.5.

The EKF forecast equations can be derived by applying a Taylor expansion to equation (2.1) at the last estimate, the analysis state $\mathbf{x}_{i-1}^a$. That is

$$\mathbf{x}_i^t = M_{i,i-1}[\mathbf{x}_{i-1}^a] + \mathbf{M}_{i,i-1}\mathbf{z}_{i-1}^a + \boldsymbol{\eta}_i + \mathcal{O}(\mathbf{z}^2) \ , \tag{2.20}$$

where $\mathbf{z}_{i-1}^a = \mathbf{x}_{i-1}^t - \mathbf{x}_{i-1}^a$ and $\mathbf{M}_{i,i-1}$ is the linearization of the operator $M_{i,i-1}$ around the estimate $\mathbf{x}_{i-1}^a$. Neglecting in equation (2.20) terms of higher than linear order in $\mathbf{z}^a$ the conditional mean and the corresponding covariance matrix of the density $p(\mathbf{x}_k^t|\mathbf{Y}_{k-\Delta k}^o)$ are computed. This yields the EKF analog of equations (2.12) and (2.13) for the forecast of the state and the forecast error covariance matrix:

$$\mathbf{x}_i^f = M_{i,i-1}[\mathbf{x}_{i-1}^a] \tag{2.21}$$

$$\mathbf{P}_k^f = \mathbf{M}_{k,k-\Delta k}\mathbf{P}_{k-\Delta k}^a\mathbf{M}_{k,k-\Delta k}^T + \mathbf{Q}_k \tag{2.22}$$

Here uncorrelated statistics of the model errors and the state were assumed as in the KF. Equation (2.21) is iterated from time $t_{k-\Delta k}$ until time t_k to obtain $\mathbf{x}_k^f$.

Since here only linear measurement operators $\mathbf{H}$ are considered, the analysis equations for the EKF are identical to those of the linear Kalman filter. Thus the analysis of the EKF is given by equations (2.15) to (2.19).

To apply the KF or EKF the filter sequence has to be initialized. That is, an initial state estimate $\mathbf{x}_0^a$ and a corresponding covariance matrix has to be supplied $\mathbf{P}_0^a$ which represent the initial probability density $p(\mathbf{x}_0^t)$.

Remark 6: The forecast of the EKF is due to linearization. The state forecast is only valid up to linear order in $\mathbf{z}$ while the covariance forecast is valid up to second order ($\mathbf{z^2} \propto \mathbf{P}^a$). The covariance matrix is forecasted by the linearized model. For nonlinear dynamics this neglect of higher order terms can lead to an unrealistic representation of the covariance matrix [39] and subsequently to instabilities of the filter algorithm [15].

Remark 7: To avoid the requirement for an adjoint model operator $\mathbf{M}_{k,k-\Delta k}^T$ the covariance forecast equation is usually applied as

$$\mathbf{P}_k^f = \mathbf{M}_{k,k-\Delta k}(\mathbf{M}_{k,k-\Delta k}\mathbf{P}_{k-\Delta k}^a)^T + \mathbf{Q}_k \tag{2.23}$$

Remark 8: The covariance matrix $\mathbf{P}$ is symmetric positive semi-definite. In a numerical implementation of the KF this property is not guaranteed to be conserved, if equation (2.17) is used to update the covariance since the operations on $\mathbf{P}$ are not symmetric. In contrast to this equation (2.16) preserves the symmetry.
Remark 9: For linear models the KF yields the optimal minimum variance estimate if the covariance matrices $\mathbf{Q}$ and $\mathbf{R}$ as well as the initial state estimate $(\mathbf{x}_0^a, \mathbf{P}_0^a)$ are correctly prescribed. Then the estimate is also the maximum likelihood estimate, see Jazwinski [38]. For nonlinear systems, the EKF can only yield an approximation of the optimal estimate. For large-scale systems, like in oceanography where the state dimension can be of order $10^5 - 10^7$, there are generally only estimates of the covariance matrices available. Also $\mathbf{x}_0^a$ is in general only an estimate of the initial system state. Due to this, the practical filter estimate is sub-optimal.
Remark 10: For large scale systems the largest computational cost resides in the forecast of the state covariance matrix by equation (2.13). This requires $2n$ applications of the (linearized) model operator. For large scale systems the corresponding computational cost is not feasible. In addition, the KF and EKF require the storage of the covariance matrix containing n^2 elements which is also not feasible for realistic models and current size of computer memory.

2.4 Error subspace Kalman Filters

The large computational cost of the KF and EKF algorithms implies that a direct application of these algorithms to realistic models with large state dimension is not feasible. This problem has led to the development of a number of approximating algorithms, sometimes called 'suboptimal schemes' after Todling and Cohn [80]. While being clearly suboptimal for linear systems, this is not necessarily true for nonlinear systems. Treating the forecast of the statistics in different manners, e.g. by nonlinear ensemble forecasts, some algorithms are better suited for application to nonlinear systems than the EKF.

This work focuses on three algorithms, the EnKF [17, 8], the SEEK Filter [68], and the SEIK Filter [67]. As far as possible the filters are presented here in the unified notation [37] following the way they have originally been introduced by the respective authors. The relation of the filters to the EKF as well as possible variations and particular features of them are discussed.

All three algorithms use a low-rank representation of the state covariance matrix $\mathbf{P}$ either by an explicit low-rank approximation of the matrix or by a random ensemble. Thus, the filter analyses operate only in a low-dimensional subspace, denoted as the error subspace. The error subspace approximates the error space considered in the EKF. It is characterized by the eigenvectors and eigenvalues of the approximated state covariance matrix. As all methods use the analysis equations of the EKF adapted to the particular method, we refer to the algorithms as Error Subspace Kalman Filters (ESKF). This corresponds to the concept of error subspace statistical estimation [49].

2.4.1 SEEK – The Singular Evolutive Extended Kalman Filter

The SEEK filter [68] is a so called reduced-rank filter. It is based on the EKF using an approximation of the covariance matrix $\mathbf{P}_0^a$ by a singular matrix of low rank and its treatment in decomposed form.

From the statistical viewpoint, the rank reduction is motivated by the fact that the probability density function $p(\mathbf{x}_0^t)$ is not isotropic in state space. If the density function is Gaussian it can be described by a probability ellipsoid, whose center is given by the mean $\mathbf{x}_0^a$ and the shape is described by $\mathbf{P}_0^a$. Figure 2.1 sketches the probability ellipsoid with its main axes in two dimensions. The principal axes of the ellipsoid are found by an eigenvalue decomposition of $\mathbf{P}_0^a$: $\{\mathbf{P}\mathbf{v}^{(i)} = \lambda^{(i)}\mathbf{v}^{(i)}, i = 1, \ldots, n\}$, where $\mathbf{v}^{(i)}$ is the i'th eigenvector and $\lambda^{(i)}$ the corresponding eigenvalue. With this, the principal vectors are $\{\tilde{\mathbf{v}}^{(i)} = (\lambda^{(i)})^{1/2}\mathbf{v}^{(i)}\}$. Approximating $\mathbf{P}_0^a$ by the r ($r \ll n$) largest eigenmodes corresponds to the neglect of the least significant principal axes of the probability ellipsoid. Also it provides the best rank-r approximation of $\mathbf{P}_0^a$, see Golub and van Loan [26]. The retained principal vectors $\{\tilde{\mathbf{v}}^{(i)},\ i = 1, \ldots, r\}$ are the basis vectors of a tangent space at the state space point $\mathbf{x}_0^a$. This is the error subspace $\tilde{\mathcal{E}}$, which approximates the true error space characterized by the full covariance matrix. The metric of $\tilde{\mathcal{E}}$ is given by $\tilde{\mathbf{G}} = \mathrm{diag}\left((\lambda^{(1)})^{-1}, \ldots, (\lambda^{(r)})^{-1}\right)$. In SEEK the error subspace is evolved until the next analysis time of the filter by forecasting the vectors $\{\mathbf{v}^{(i)},\ i = 1, \ldots, r\}$ with the linearized model. In the analysis phase the filter operates only in the error subspace, that is, in the most significant directions of uncertainty.

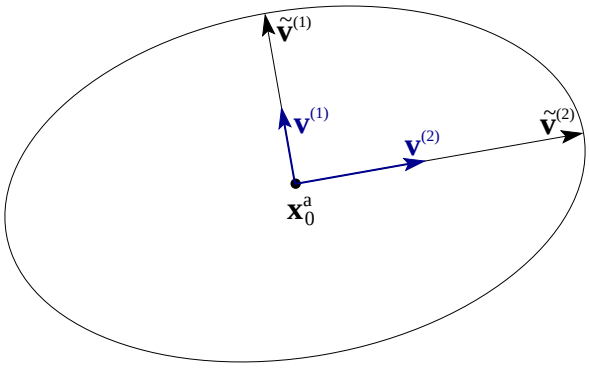


Figure 2.1: Probability ellipsoid representing the probability density function $p(\mathbf{x}_0^t)$.

The SEEK filter is given by the following equations:

Initialization:

The initial probability density $p(\mathbf{x}_0^t)$ is provided by the initial state estimate $\mathbf{x}_0^a$ and a rank-r approximation ($r \ll n$) of the covariance matrix $\mathbf{P}_0^a$ given in decomposed form:

$$\mathbf{x}_0^a =< \mathbf{x}_0^t >; \quad \hat{\mathbf{P}}_0^a := \mathbf{V}_0\mathbf{U}_0\mathbf{V}_0^T \approx \mathbf{P}_0^a \tag{2.24}$$

Here the diagonal matrix $\mathbf{U}_0 \in \mathbb{R}^{r\times r}$ holds the r largest eigenvalues. Matrix $\mathbf{V}_0 \in \mathbb{R}^{n\times r}$ contains in its columns the corresponding eigenvectors (modes) of $\hat{\mathbf{P}}_0^a$.

Forecast:
The SEEK forecast equations are derived from the EKF by treating the covariance matrix in decomposed form as provided by the initialization.

$$\mathbf{x}_i^f = M_{i,i-1}[\mathbf{x}_{i-1}^a] \tag{2.25}$$
$$\mathbf{V}_k = \mathbf{M}_{k,k-\Delta k}\mathbf{V}_{k-\Delta k} \tag{2.26}$$

Analysis:
The analysis equations are a re-formulation of the EKF analysis equations for a covariance matrix given in decomposed form. To maintain the rank r of $\hat{\mathbf{P}}_0^a$ the model error covariance matrix $\mathbf{Q}_k$ is projected onto the error subspace by

$$\hat{\mathbf{Q}}_k := \left(\mathbf{V}_k^T\mathbf{V}_k\right)^{-1}\mathbf{V}_k^T\mathbf{Q}_k\mathbf{V}_k\left(\mathbf{V}_k^T\mathbf{V}_k\right)^{-1} \ . \tag{2.27}$$

With this the SEEK analysis equations are for an invertible matrix $\mathbf{R}_k$

$$\mathbf{U}_k^{-1} = \left(\mathbf{U}_{k-\Delta k} + \hat{\mathbf{Q}}_k\right)^{-1} + \left(\mathbf{H}_k\mathbf{V}_k\right)^T\mathbf{R}_k^{-1}\mathbf{H}_k\mathbf{V}_k \ , \tag{2.28}$$
$$\mathbf{x}_k^a = \mathbf{x}_k^f + \hat{\mathbf{K}}_k\left(\mathbf{y}_k^o - \mathbf{H}_k\mathbf{x}_k^f\right) \ , \tag{2.29}$$
$$\hat{\mathbf{K}}_k = \mathbf{V}_k\mathbf{U}_k\mathbf{V}_k^T\mathbf{H}_k^T{\mathbf{R}_k}^{-1} \ . \tag{2.30}$$

The analysis covariance matrix is implicitly given by $\hat{\mathbf{P}}_k^a := \mathbf{V}_k\mathbf{U}_k\mathbf{V}_k^T$.
Re-diagonalization:
To avoid that the modes $\{\mathbf{v}_{(i)}\}$ become large and increasingly aligned a re-orthonormalization of these vectors is required. This can be performed by computing the eigenvalue decomposition of the matrix $\mathbf{B}_k \in \mathbb{R}^{r\times r}$ defined by

$$\mathbf{B}_k := \mathbf{A}_k^T\mathbf{V}_k^T\mathbf{V}_k\mathbf{A}_k \tag{2.31}$$

where $\mathbf{A}_k$ is computed by a Cholesky decomposition of the matrix $\mathbf{U}_k$: $\mathbf{U}_k = \mathbf{A}_k\mathbf{A}_k^T$. The eigenvalues of $\mathbf{B}_k$ are the same as the non-zero eigenvalues of $\mathbf{P}_k^a = \mathbf{V}_k\mathbf{U}_k\mathbf{V}_k^T$. Let $\mathbf{C}_k$ contain in its columns the eigenvectors of $\mathbf{B}_k$ and the diagonal matrix $\mathbf{D}_k$ the corresponding eigenvalues. Then the matrix $\tilde{\mathbf{V}}$ holding re-orthonormalized modes and the corresponding eigenvalue matrix $\hat{\mathbf{U}}$ are given by

$$\hat{\mathbf{V}}_k = \mathbf{L}_k\mathbf{C}_k\mathbf{D}_k^{-1/2} \ ; \quad \hat{\mathbf{U}}_k = \mathbf{D}_k \ . \tag{2.32}$$

Remark 11: The state covariance matrix is approximated by a singular matrix $\hat{\mathbf{P}}$ of low rank. Throughout the algorithm the approximated matrix is treated in the decomposed form $\hat{\mathbf{P}} = \mathbf{V}\mathbf{U}\mathbf{V}^T$. The full covariance matrix is never computed explicitly and has never to be stored.
Remark 12: Due to its treatment in decomposed form, all operations on $\hat{\mathbf{P}}$ are performed symmetrically. Hence, $\hat{\mathbf{P}}$ remains symmetric throughout the algorithm.
Remark 13: It is not required that the decomposition of $\hat{\mathbf{P}}$ is computed from a truncated eigenvalue decomposition of the prescribed matrix $\mathbf{P}_0^a$. However, mathematically this yields the best approximation of $\mathbf{P}_0^a$.

Remark 14: The forecast of the covariance matrix is computed by only forecasting the r modes of $\hat{\mathbf{P}}$. With typically $r < 100$ this brings this forecast toward acceptable computation times.
Remark 15: The SEEK filter is a re-formulation of the EKF focusing on the analyzed state estimate and covariance matrix. Hence its filtering performance will be sub-optimal. Further, SEEK inherits the stability problem of the EKF by considering only the two lowest statistical moments of the probability density. If r is too small, this problem is even amplified, as $\hat{\mathbf{P}}^a$ systematically underestimates the variance prescribed by the full covariance matrix $\mathbf{P}^a$. This is due to the neglect of eigenvalues of the positive semi-definite matrix $\mathbf{P}^a$.
Remark 16: The increment for the analysis update of the state estimate in equation (2.29) is computed as a weighted average of the mode vectors in $\mathbf{V}_k$ which belong to the error subspace. This becomes visible when the definition of the Kalman gain (equation (2.30)) is inserted into equation (2.29):

$$\mathbf{x}_k^a = \mathbf{x}_k^f + \mathbf{V}_k \left[\mathbf{U}_k \mathbf{V}_k^T \mathbf{H}_k^T \mathbf{R}_k{}^{-1} \left(\mathbf{y}_k^o - \mathbf{H}_k \mathbf{x}_k^f \right) \right] \tag{2.33}$$

The term in brackets represents a vector of weights for combining the modes $\mathbf{V}$.
Remark 17: In practice, it can be difficult to specify the linearized dynamic model operator $\mathbf{M}_{i,i-1}$. As an alternative, one can approximate the linearization by a gradient approximation. Then, the forecast of column α of $\mathbf{V}_{i-1}^a$, denoted by $\mathbf{v}_{i-1}^{a(\alpha)}$, is given by

$$\mathbf{M}_{i,i-1} \mathbf{v}_{i-1}^{a(\alpha)} \approx \frac{M_{i,i-1}[\mathbf{x}_{i-1}^a + \epsilon \mathbf{v}_{i-1}^{a(\alpha)}] - M_{i,i-1}[\mathbf{x}_{i-1}^a]}{\epsilon} \,. \tag{2.34}$$

For a gradient approximation the coefficient ϵ needs to be a small positive number ($\epsilon \ll 1$). Some authors [91, 31] report the use of $\epsilon \approx 1$. This can bring the algorithm beyond a purely tangent-linear forecast but it is no more defined as a gradient approximation and requires an ensemble interpretation.
Remark 18: Due the neglect of higher order terms in the Taylor expansion (2.20) the forecast of the state estimate will be systematically biased. To account for the first neglected term in the Taylor expansion second order forecast schemes have been discussed [87, 73]. The examination of the forecast bias can also be utilized to quantify the nonlinearity of the forecast [89].
Remark 19: Equation (2.28) for the matrix $\mathbf{U}_k$ can be modified by multiplying with a so called forgetting factor ρ, $(0 < \rho \leq 1)$ [68]:

$$\mathbf{U}_k^{-1} = (\rho^{-1} \mathbf{U}_{k-\Delta k} + \hat{\mathbf{Q}}_k)^{-1} + (\mathbf{H}_k \mathbf{V}_k)^T \mathbf{R}_k^{-1} \mathbf{H}_k \mathbf{V}_k \tag{2.35}$$

The forgetting factor can be used as a tuning parameter of the analysis phase to down-weight the state forecast relative to the observations. This can increase the filter stability as the systematic underestimation of the variance is reduced.
Remark 20: In equation (2.26) the modes $\mathbf{V}$ of $\hat{\mathbf{P}}$ are evolved with initially unit norm in the state space. However, it is also possible to use modes scaled by the square root of the corresponding eigenvalue, i.e. the basis vectors of the error subspace, Then, matrix $\mathbf{U}$ will be the identity matrix. Using scales modes the re-diagonalization should

be performed after each analysis stage, replacing equations (2.32) by $\hat{\mathbf{V}}_k = \mathbf{V}_k\mathbf{C}_k$ and $\hat{\mathbf{U}}_k = \mathbf{I}_{r\times r}$. This scaled algorithm is equivalent to the RRSQRT algorithm introduced by Verlaan and Heemink [88].

2.4.2 EnKF – The Ensemble Kalman Filter

The EnKF [17, 8] applies a Monte Carlo method to sample and forecast the probability density function. The initial density $p(\mathbf{x}_0^t)$ is sampled by a finite random ensemble of state realizations. The density is forecasted by evolving each ensemble member with the full stochastic model. For the analysis each ensemble state is updated using an observation vector from an ensemble of observations, which has to be generated according to the observation error covariance matrix.

From the viewpoint of statistics the EnKF solves the Fokker-Planck-Kolmogorov equation (2.3) for the evolution of the probability density $p(\mathbf{x}^t)$ by a Monte Carlo method. In contrast to the SEEK algorithm, where the rank reduction directly uses the assumption that the density is Gaussian and thus can be described by a probability ellipsoid, the EnKF samples the density by a random ensemble of N model states $\{\mathbf{x}_0^{a(\alpha)},\ \alpha = 1,\dots,N\}$. The probability density is given in terms of the ensemble member density in state space dN:

$$\frac{dN}{N} \rightarrow p(\mathbf{x}_0^t)d\mathbf{x} \quad \text{for } N\rightarrow\infty \tag{2.36}$$

This sampling of $p(\mathbf{x}_0^t)$ converges rather slow (proportional to $N^{-1/2}$), but it is valid for any kind of probability density, not just Gaussian densities. Forecasting each ensemble state with the stochastic-dynamic model (2.1) evolves the sampled density with the nonlinear model until the next analysis time. In the analysis phase the EKF analysis, which implies that the densities are Gaussian, is applied to each of the ensemble states. For the analysis the covariance matrix $\mathbf{P}$ is approximated by the ensemble covariance matrix $\tilde{\mathbf{P}}$. Since the rank of $\tilde{\mathbf{P}}$ is at most $N-1$, the EnKF also operates in an error subspace which is determined by the random sampling. Unlike the SEEK filter the directions are not provided by the principal vectors of the prescribed covariance matrix but determined by the random sampling. To ensure that the ensemble analysis represents the combination of two probability densities, the observation error covariance matrix $\mathbf{R}$ has to be represented by a random ensemble of observations [8]. Each ensemble state is then updated with a vector from this observation ensemble. This implicitly updates the state covariance matrix.

The EnKF algorithm is prescribed by the following equations:
Initialization:
The initial probability density $p(\mathbf{x}_0^t)$ is sampled by a random ensemble of N state realizations. The statistics of this ensemble approximate the initial state estimate and the corresponding covariance matrix, thus

$$\{\mathbf{x}_0^{a(\alpha)}, \alpha = 1,\dots,N\} \tag{2.37}$$

with

$$\overline{\mathbf{x}_0^a} = \frac{1}{N}\sum_{\alpha=1}^{N}\mathbf{x}_0^{a(\alpha)} \rightarrow < \mathbf{x}_0^t > \text{ for } N \rightarrow \infty, \tag{2.38}$$

$$\tilde{\mathbf{P}}_0^a := \frac{1}{N-1}\sum_{\alpha=1}^{N}\left(\mathbf{x}_0^{a(\alpha)} - \overline{\mathbf{x}_0^a}\right)\left(\mathbf{x}_0^{a(\alpha)} - \overline{\mathbf{x}_0^a}\right)^T \rightarrow \mathbf{P}_0^a \quad \text{for } N \rightarrow \infty. \tag{2.39}$$

Forecast:
Each ensemble member is evolved up to time t_k with the nonlinear stochastic-dynamic model (2.1) as

$$\mathbf{x}_i^{f(\alpha)} = M_{i,i-1}[\mathbf{x}_{i-1}^{a(\alpha)}] + \boldsymbol{\eta}_i^{(\alpha)} \ . \tag{2.40}$$

Analysis:
For the analysis a random ensemble of observation vectors $\{\mathbf{y}_k^{o(\beta)},\ \beta = 1,\ldots,N\}$ is generated which represents an approximate observation error covariance matrix ($\tilde{\mathbf{R}}_k \approx \mathbf{R}_k$). Each of the ensemble members is updated analogously to the EKF analysis by

$$\mathbf{x}_k^{a(\alpha)} = \mathbf{x}_k^{f(\alpha)} + \tilde{\mathbf{K}}_k\left(\mathbf{y}_k^{o(\alpha)} - \mathbf{H}_k\mathbf{x}_k^{f(\alpha)}\right) , \tag{2.41}$$

$$\tilde{\mathbf{K}}_k = \tilde{\mathbf{P}}_k^f\mathbf{H}_k^T\left(\mathbf{H}_k\tilde{\mathbf{P}}_k^f\mathbf{H}_k^T + \mathbf{R}_k\right)^{-1} , \tag{2.42}$$

$$\tilde{\mathbf{P}}_k^f = \frac{1}{N-1}\sum_{\alpha=1}^{N}\left(\mathbf{x}_k^{f(\alpha)} - \overline{\mathbf{x}_k^f}\right)\left(\mathbf{x}_k^{f(\alpha)} - \overline{\mathbf{x}_k^f}\right)^T . \tag{2.43}$$

The analysis state and corresponding covariance matrix are then defined by the ensemble mean and covariance matrix as

$$\mathbf{x}_k^a := \frac{1}{N}\sum_{\alpha=1}^{N}\mathbf{x}_k^{a(\alpha)} , \tag{2.44}$$

$$\tilde{\mathbf{P}}_k^a := \frac{1}{N-1}\sum_{\alpha=1}^{N}\left(\mathbf{x}_k^{a(\alpha)} - \mathbf{x}_k^a\right)\left(\mathbf{x}_k^{a(\alpha)} - \mathbf{x}_k^a\right)^T \tag{2.45}$$

which complete the analysis equations of the EnKF.

An efficient implementation of this analysis is formulated in terms of "representers" [19]. This formulation as well permits to handle the situation when $\mathbf{H}_k\tilde{\mathbf{P}}_k^f{\mathbf{H}_k}^T$ is singular, which will occur if $m_k > N$. The state analysis equation (2.41) is written as

$$\mathbf{x}_k^{a(\alpha)} = \mathbf{x}_k^{f(\alpha)} + \tilde{\mathbf{P}}_k^f\mathbf{H}_k^T\mathbf{b}_k^{(\alpha)} \ . \tag{2.46}$$

The columns of the matrix $\tilde{\mathbf{P}}_k^f\mathbf{H}_k^T$ are called representers and constitute influence vectors for each of the measurements. Amplitudes for the influence vectors are given by the vectors $\{\mathbf{b}_k^{(\alpha)}\}$ which are obtained as the solution of

$$\left(\mathbf{H}_k\tilde{\mathbf{P}}_k^f{\mathbf{H}_k}^T + \mathbf{R}_k\right)\mathbf{b}_k^{(\alpha)} = \mathbf{y}_k^{o(\alpha)} - \mathbf{H}_k\mathbf{x}_k^{f(\alpha)} \ . \tag{2.47}$$

The explicit computation of $\tilde{\mathbf{P}}_k^f$ by equation (2.43), is not required in the algorithm. It suffices to compute (see, for example Houtekamer and Mitchell [34])

$$\tilde{\mathbf{P}}_k^f \mathbf{H}_k^T = \frac{1}{N-1} \sum_{\alpha=1}^{N} \left(\mathbf{x}_k^{f(\alpha)} - \overline{\mathbf{x}_k^f} \right) \left[\mathbf{H}_k \left(\mathbf{x}_k^{f(\alpha)} - \overline{\mathbf{x}_k^f} \right) \right]^T , \tag{2.48}$$

$$\mathbf{H}_k \tilde{\mathbf{P}}_k^f \mathbf{H}_k^T = \frac{1}{N-1} \sum_{\alpha=1}^{N} \mathbf{H}_k \left(\mathbf{x}_k^{f(\alpha)} - \overline{\mathbf{x}_k^f} \right) \left[\mathbf{H}_k \left(\mathbf{x}_k^{f(\alpha)} - \overline{\mathbf{x}_k^f} \right) \right]^T . \tag{2.49}$$

For later use we also introduce the matrix notation of the EnKF. The initial state ensemble matrix holds in its columns the ensemble state as as $\mathbf{X}_0^a = \{\mathbf{x}_0^{a(1)}, \ldots, \mathbf{x}_0^{a(N)}\}$. Introducing the ensemble matrix of the observation vectors $\mathbf{Y}_k^o = \{\mathbf{y}_k^{o(1)}, \ldots, \mathbf{y}_k^{o(N)}\}$ we can rewrite equation (2.47) for the influence amplitudes as

$$\left(\mathbf{H}_k \tilde{\mathbf{P}}_k^f {\mathbf{H}_k}^T + \mathbf{R}_k \right) \mathbf{B}_k = \mathbf{Y}_k^o - \mathbf{H}_k \mathbf{X}_k^f \tag{2.50}$$

where $\mathbf{B}_k$ is the matrix of influence amplitudes. The ensemble update (equation 2.46) is now given as

$$\mathbf{X}_k^a = \mathbf{X}_k^f + \tilde{\mathbf{P}}_k^f \mathbf{H}_k^T \mathbf{B}_k . \tag{2.51}$$

In addition, the computation of the representers $\tilde{\mathbf{P}}_k^f \mathbf{H}_k^T$ and the covariance matrix $\mathbf{H}_k \tilde{\mathbf{P}}_k^f \mathbf{H}_k^T$ is written in matrix notation as

$$\tilde{\mathbf{P}}_k^f \mathbf{H}_k^T = \frac{1}{N-1} \left(\mathbf{X}_k^f - \overline{\mathbf{X}_k^f} \right) \left[\mathbf{H}_k \left(\mathbf{X}_k^f - \overline{\mathbf{X}_k^f} \right) \right]^T , \tag{2.52}$$

$$\mathbf{H}_k \tilde{\mathbf{P}}_k^f \mathbf{H}_k^T = \frac{1}{N-1} \mathbf{H}_k \left(\mathbf{X}_k^f - \overline{\mathbf{X}_k^f} \right) \left[\mathbf{H}_k \left(\mathbf{X}_k^f - \overline{\mathbf{X}_k^f} \right) \right]^T . \tag{2.53}$$

Here the matrix $\overline{\mathbf{X}_k^f}$ contains in all columns the vector $\overline{\mathbf{x}_k^f}$.

The EnKF comprises some particular features due to the use of a Monte Carlo method in all phases of the filter:

Remark 21: The EnKF treats the covariance matrix implicitly in a square root form, as is evident from equations (2.43) and (2.45). With this the covariance matrix remains symmetric in the EnKF. As in the SEEK algorithm it is neither required to store the full covariance matrix nor to compute it explicitly.

Remark 22: The forecast phase evolves all N ensemble states with the nonlinear model. This also allows for non-Gaussian densities. Algorithmically the ensemble evolution has the benefit that a linearized model operator is not required.

Remark 23: The analysis phase is derived from the EKF. Thus, it only accounts for the two lowest statistical moments of the probability density. Using the mean of the forecast ensemble as state forecast estimate leads for sufficiently large ensembles to a more accurate estimate than in the EKF. From the Taylor expansion, equation (2.20), it is obvious that this takes into account higher order terms than the EKF does. In contrast to the EKF and SEEK filters $\mathbf{P}$ is only updated implicitly by the analysis of the ensemble states.

Remark 24: The representer analysis method applied in the EnKF operates on the observation space. Hence, the error subspace is not explicitly considered. An algorithm which operates on the error subspace is given by the concept of Error Subspace Statistical Estimation (ESSE) [49].
Remark 25: The analysis increments for the ensemble states are computed as weighted means of the vectors $\mathbf{X}_k^f - \overline{\mathbf{X}_k^f}$ which belong to the error subspace. Thus the analysis equation (2.51) for the ensemble update can be written as

$$\mathbf{X}_k^a = \mathbf{X}_k^f + \left(\mathbf{X}_k^f - \overline{\mathbf{X}_k^f}\right)\left(\frac{1}{N-1}\left[\mathbf{H}_k\left(\mathbf{X}_k^f - \overline{\mathbf{X}_k^f}\right)\right]^T \mathbf{B}_k\right) \tag{2.54}$$

Evensen [18] noted that the analysis can also be interpreted as a weakly nonlinear combination of the ensemble states. The first interpretation, however, shows that the update increments are computed in the error subspace.
Remark 26: Using a Monte-Carlo sampling of the initial probability density also non-Gaussian densities can be represented. As the sampling convergences slowly with $\mathcal{O}(N^{-1/2})$, rather large ensembles ($N \geq 100$) are required [17, 19] to avoid too big sampling errors.
Remark 27: To enhance the quality of the filter estimate for small ensemble sizes a variant of the EnKF has been proposed which uses a pair of ensembles [34]. From the mathematical viewpoint it is, however, advisable to use as large as possible ensembles to ensure that the statistics can be estimated correctly. In addition, for a given ensemble size the state estimate of the EnKF using a single ensemble is better than the state estimate of the double-ensemble EnKF with the same total number of ensemble states [84, 35].
Remark 28: Since the estimated correlations of the EnKF will be noisy for small ensembles it has been proposed [36] to filter the covariances by a Schur product of correlations functions of local support with the ensemble covariance matrix. This technique filters out noisy long-range correlations. Further, correlations are intermediate distances will be weakened. Hence, the influence of observations are intermediate distances is reduced, see [30]. The localization will, however, introduce imbalances into the ensemble states as has been studied by Mitchell et al. [56].
Remark 29: The generation of an observation ensemble is required to ensure consistent statistics of the updated state ensemble [8]. With the observation ensemble the covariance matrix $\mathbf{R}_k$ is represented as $\tilde{\mathbf{R}}_k$ in equation (2.16) which would be missing otherwise. This, however, introduces additional sampling error to the ensemble which is largest when the ensemble is small compared with the rank of $\mathbf{R}_k$, e.g. if $\mathbf{R}_k$ is diagonal. Furthermore, it is likely that the state and observation ensembles have spurious correlations. This introduces an additional error term in equation (2.16).
Remark 30: In equations (2.42) and (2.47) it is possible to use, instead of the prescribed matrix $\mathbf{R}_k$, the ensemble error covariance matrix $\tilde{\mathbf{R}}_k$ of the observation ensemble $\{\mathbf{y}_k^{o(\beta)},\ k = 1, \ldots, N\}$. As proposed by Evensen [18], this allows for an analysis scheme which is numerically very efficient. However, due to the sampling problems of $\mathbf{R}_k$ this can lead to a further degradation of the filter quality.
Remark 31: To avoid the requirement of an ensemble of observations, several algorithms have been proposed which perform the analysis only on the ensemble mean and

transform the ensemble after this update [1, 5, 94]. These filter algorithms can be interpreted in a unified way as ensemble square root filters [79].

2.4.3 SEIK – The Singular Evolutive Interpolated Kalman Filter

The SEIK filter [67] has been derived as a variant of the SEEK algorithm. It uses interpolation instead of linearization for the forecast phase. Alternatively it can be interpreted as an ensemble Kalman filter using a preconditioned ensemble. As in the SEEK algorithm the SEIK filter uses a low-rank approximation of the covariance matrix. From this an ensemble of minimum size is generated whose ensemble statistics exactly reproduce the approximated covariance matrix. The ensemble is forecasted with the nonlinear model like in the EnKF algorithm. The analysis is performed in analogy to that of the SEEK filter with a single observation vector using the ensemble mean and covariance matrix. Subsequent to the analysis, the state ensemble is resampled to represent the analysis state estimate and covariance matrix. The SEIK algorithm should not be confused with other interpolated variants of the SEEK filter, e.g. [90], which typically correspond to the SEEK filter with gradient approximation.

Statistically the initialization of the SEIK filter is analogous to that of the SEEK: The probability density $p(\mathbf{x}_0^t)$ is again represented by the principal axes of $\mathbf{P}_0^a$ and approximated by the r largest eigenmodes. In the SEIK algorithm the eigenmodes are, however, not directly evolved but a random ensemble of $r+1$ state realizations is generated. This ensemble exactly represents the mean and covariance matrix of the approximated probability density. The density is forecasted by evolving each of the ensemble members with the nonlinear model. The evolved error subspace is determined by computing the forecast state estimate and covariance matrix from the ensemble. The analysis is performed analogous to the SEEK filter. This Kalman-type analysis assumes again Gaussian densities.

The SEIK filter is given by the following equations:
Initialization:
The initial probability density $p(\mathbf{x}_0^t)$ is provided by the initial state estimate $\mathbf{x}_0^a$ and a rank-r approximation of $\mathbf{P}_0^a$ given in decomposed form as

$$\mathbf{x}_0^a =< \mathbf{x}_0^t >; \quad \hat{\mathbf{P}}_0^a := \mathbf{V}_0\mathbf{U}_0\mathbf{V}_0^T \approx \mathbf{P}_0^a \ . \tag{2.55}$$

From this information an ensemble of $r+1$ state realizations is generated as the state matrix

$$\mathbf{X}_0^a = \{\mathbf{x}_0^{a(1)}, \ldots, \mathbf{x}_0^{a(r+1)}\} \tag{2.56}$$

with

$$\overline{\mathbf{x}_0^a} \equiv \mathbf{x}_0^a \ , \tag{2.57}$$

$$\check{\mathbf{P}}_0^a := \frac{1}{r+1}\sum_{\alpha=1}^{r+1}(\mathbf{x}_0^{a(\alpha)} - \overline{\mathbf{x}_0^a})(\mathbf{x}_0^{a(\alpha)} - \overline{\mathbf{x}_0^a})^T \equiv \hat{\mathbf{P}}_0^a \ . \tag{2.58}$$

To ensure that equations (2.57) and (2.58) hold, the ensemble is generated in a procedure called minimum second-order exact sampling [65][2]. For this, let $\mathbf{C}_0$ contain in its diagonal the square roots of the eigenvalues of $\hat{\mathbf{P}}_0^a$, such that $\mathbf{U}_0 = \mathbf{C}_0^T\mathbf{C}_0$. Then $\check{\mathbf{P}}_0^a$ is written as

$$\check{\mathbf{P}}_0^a = \mathbf{V}_0\mathbf{C}_0^T\mathbf{\Omega}_0^T\mathbf{\Omega}_0\mathbf{C}_0\mathbf{V}_0^T \ , \tag{2.59}$$

where $\mathbf{\Omega}_0$ is a $(r+1)\times r$ random matrix whose columns are orthonormal and orthogonal to the vector $(1,\ldots,1)^T$ which can be obtained by Householder reflections, see e.g. Hoteit et al. [33]. The state realizations of the ensemble are then given by

$$\mathbf{x}_0^{a(\alpha)} = \mathbf{x}_0^a + \sqrt{r+1}\ \mathbf{V}_0\mathbf{C}_0^T(\mathbf{\Omega}_0^T)^{(\alpha)} \ , \tag{2.60}$$

where $(\mathbf{\Omega}_0^T)^{(\alpha)}$ denotes the α-th column of $\mathbf{\Omega}_0^T$.

$\check{\mathbf{P}}_0^a$ can also be described in terms of the ensemble states by

$$\check{\mathbf{P}}_0^a = \frac{1}{r+1}\mathbf{X}_0^a\mathbf{T}(\mathbf{T}^T\mathbf{T})^{-1}\mathbf{T}^T(\mathbf{X}_0^a)^T \ . \tag{2.61}$$

$\mathbf{T}$ is a $(r+1)\times r$ matrix with zero column sums. A possible choice for $\mathbf{T}$ is

$$\mathbf{T} = \begin{pmatrix} \mathbf{I}_{r\times r} \\ \mathbf{0}_{1\times r} \end{pmatrix} - \frac{1}{r+1}\left(\mathbf{1}_{(r+1)\times r}\right) \ . \tag{2.62}$$

Here $\mathbf{0}$ represents the matrix whose elements are equal to zero. The elements of the matrix $\mathbf{1}$ are equal to one. Matrix $\mathbf{T}$ fulfills the purpose of implicitly subtracting the ensemble mean when computing $\check{\mathbf{P}}_0^a$. Equation (2.61) can be written in a form analogous to the covariance matrix in (2.55) as

$$\check{\mathbf{P}}_0^a = \mathbf{L}_0\mathbf{G}\mathbf{L}_0^T \tag{2.63}$$

with

$$\mathbf{L}_0 \ := \ \mathbf{X}_0^a\,\mathbf{T} \ , \tag{2.64}$$

$$\mathbf{G} \ := \ \frac{1}{r+1}\left(\mathbf{T}^T\mathbf{T}\right)^{-1} \ . \tag{2.65}$$

Forecast:
Each ensemble member is evolved up to time t_k with the nonlinear dynamic model equation

$$\mathbf{x}_i^{f(\alpha)} = M_{i,i-1}[\mathbf{x}_{i-1}^{a(\alpha)}] \ . \tag{2.66}$$

Analysis:
The analysis equations are analogous to the SEEK filter, but here the forecast state estimate is given by the ensemble mean $\overline{\mathbf{x}_k^f}$. To maintain the rank r of $\check{\mathbf{P}}_k$ matrix $\mathbf{Q}_k$ is again projected onto the error subspace according to equation (2.27) with $\mathbf{V}_k$

[2]Note that the definitions of the sampled covariance matrices are different in EnKF and SEIK. The EnKF uses a normalization factor $(N-1)^{-1}$ while SEIK uses $(r+1)^{-1} = N^{-1}$. However, in both algorithms the ensemble is generated to be consistent with the respective definition of the covariance matrix.

replaced by $\mathbf{L}_k$ defined by equation (2.64). $\mathbf{U}_k$ is updated as in the SEEK algorithm (equation (2.28)), but with $\mathbf{U}_{k-\Delta k}$ being replaced by the constant matrix $\mathbf{G}$ (equation 2.65). Thus, the analysis equations are

$$\begin{aligned} \mathbf{U}_k^{-1} &= [\mathbf{G} + \check{\mathbf{Q}}_k]^{-1} + (\mathbf{H}_k\mathbf{L}_k)^T\mathbf{R}_k^{-1}\mathbf{H}_k\mathbf{L}_k \ , & (2.67)\\ \mathbf{x}_k^a &= \overline{\mathbf{x}_k^f} + \check{\mathbf{K}}_k(\mathbf{y}_k^o - \mathbf{H}_k\overline{\mathbf{x}_k^f}) \ , & (2.68)\\ \check{\mathbf{K}}_k &= \mathbf{L}_k\mathbf{U}_k\mathbf{L}_k^T\mathbf{H}_k^T{\mathbf{R}_k}^{-1} \ . & (2.69) \end{aligned}$$

The analysis covariance matrix is implicitly given by $\check{\mathbf{P}}_k^a := \mathbf{L}_k\mathbf{U}_k\mathbf{L}_k^T$.

Resampling:

To proceed with the filter sequence the ensemble has to be resampled in consistency with relations (2.57) and (2.58) at time t_k. The procedure is analogous to the initial ensemble generation but here a Cholesky decomposition is applied to obtain $\mathbf{U}_k^{-1} = \mathbf{C}_k\mathbf{C}_k^T$. Then $\check{\mathbf{P}}_k^a$ can be written in analogy to (2.59) as

$$\check{\mathbf{P}}_k^a = \mathbf{L}_k(\mathbf{C}_k^{-1})^T\mathbf{\Omega}_k^T \ \mathbf{\Omega}_k\mathbf{C}_k^{-1}\mathbf{L}_k^T \ , \tag{2.70}$$

where $\mathbf{\Omega}_k$ has the same properties as in the initialization. Accordingly the ensemble members are given by

$$\mathbf{x}_k^{a(\alpha)} = \mathbf{x}_k^a + \sqrt{r+1}\ \mathbf{L}_k(\mathbf{C}_k^{-1})^T(\mathbf{\Omega}_k^T)^{(\alpha)} \ . \tag{2.71}$$

The SEIK algorithm shares features of both the SEEK and the EnKF filters:

Remark 32: Using second order exact sampling of the low-rank approximated covariance matrix leads to smaller sampling errors of the ensemble covariance matrix compared with the Monte Carlo sampling in the EnKF.

Remark 33: The ensemble members are evolved with the nonlinear model. Thus, as algorithmic benefit, the linearized model operator is not required. In addition, the nonlinear ensemble evolution yields a more realistic forecast of the covariance matrix compared with the SEEK filter. Furthermore, the forecast permits to treat model errors as a stochastic forcing like in the EnKF.

Remark 34: The forecast state estimate is computed as the mean of the ensemble forecast. Analogous to the EnKF this leads to a forecast accounting for higher order terms in the Taylor expansion equation (2.20).

Remark 35: Like in the SEEK filter, the analysis phase of the SEIK operates only in an error subspace given by the most significant directions of uncertainty. With this the SEIK filter is analogous to the concept of Error Subspace Statistical Estimation (ESSE) [49]. The difference of the SEIK to square root EnKF algorithms [1, 5, 94, 79] lies in the fact that these algorithms compute the analysis update in the observation space rather than the error subspace.

Remark 36: The forecast phase uses an ensemble which exactly represents the low-rank approximated state covariance matrix. It has the minimal size $r + 1$. A similar scheme, called unscented transformation, has been discussed by Julier et al. [40, 39]. This scheme evolves an ensemble of $2r + 1$ states. The ensemble is initialized by the state estimate $\mathbf{x}_0^a$, the r states $\{\mathbf{x}_0^a + \tilde{\mathbf{v}}^{(\alpha)},\ \alpha = 1, \ldots, r\}$, and the r states $\{\mathbf{x}_0^a - \tilde{\mathbf{v}}^{(\alpha)}\}$ where the $\{\tilde{\mathbf{v}}^{(\alpha)}\}$ are the basis vectors of the error subspace.

2.5 Nonlinear Measurement Operators

We formulated the Kalman filter and the error subspace Kalman filters with linear measurement operators $\mathbf{H}_k$. It is, in general possible to apply nonlinear measurement operators H_k with these filters. As we will explain below, the application of a nonlinear measurement operator cannot be expected to provide an optimal filter estimate.

2.5.1 Nonlinear Measurement Operators in the Extended Kalman Filter

To derive the EKF analysis equations with a nonlinear measurement operator a Taylor expansion is applied to the observation model (2.2) at the forecast state $\mathbf{x}_k^f$. Writing $\mathbf{z}_k^f := \mathbf{x}_k^t - \mathbf{x}_k^f$ it is

$$\mathbf{y}_k^o = H_k[\mathbf{x}_k^f] + \mathbf{H}_k\mathbf{z}_k^f + \boldsymbol{\epsilon}_k + \mathcal{O}(\mathbf{z}^2) \ . \tag{2.72}$$

Here $\mathbf{H}_k$ is the linearization of the measurement operator H_k around the forecast estimate $\mathbf{x}_k^f$. Neglecting in the expansion terms of higher than linear order in $\mathbf{z}_k^f$, the analysis equations with nonlinear H are obtained analogous to equations (2.15) to (2.18) as

$$\begin{aligned} \mathbf{x}_k^a &= \mathbf{x}_k^f + \mathbf{K}_k(\mathbf{y}_k^o - H_k[\mathbf{x}_k^f]) \ , & (2.73)\\ \mathbf{P}_k^a &= (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_k^f & (2.74) \end{aligned}$$

The Kalman gain $\mathbf{K}_k$ is again given by equation (2.18).

The problem in the application of nonlinear measurement operators lies in the fact that the derivation of the analysis equations of the KF implicitly assumes that $\mathbf{H}_k\mathbf{x}_K^t$ is Gaussian distributed. If the distribution of $\mathbf{x}_k^t$ is Gaussian this will be fulfilled for a linear operator $\mathbf{H}_k$. However, the nonlinear transformation $H_k[\mathbf{x}_k^t]$ will not yield a Gaussian distribution, even if $\mathbf{x}_k^t$ is Gaussian. Due to this, the analysis probability density $p(\mathbf{x}_k^t|\mathbf{Y}_k^o)$ will not be Gaussian and hence not be completely described by its mean and covariance matrix. Hence, the filter estimate will be sub-optimal for all filters which are based on the analysis equations of the Kalman filter. The state estimate will not be the minimum variance estimate. In some situations, this can yield stability problems, as was shown, e.g., by van Leeuwen [85]. A possible, more consistent, way to cope with the nonlinear H is to apply an iterative analysis scheme instead of the EKF analysis equations (2.73) and (2.74), see e.g. [38, 11].

2.5.2 Direct Application of Nonlinear Measurement Operators

Despite the fact that nonlinear measurement operators will yield a sub-optimal filter estimate, there is no reason which would forbid their application at all. In the error subspace filter algorithms which use an ensemble formulation, namely the SEIK and the EnKF algorithm, the nonlinear measurement operators can be directly applied. We discuss this first in the context of the EnKF algorithm as has been shown e.g. by

Houtekamer and Mitchell [36]. Since all fields and operators refer to the time t_k the time index is omitted in the following.

The application of the nonlinear operator H is, in general, always valid when applied to a real model state $\mathbf{x}$. Due to the nonlinearity the application of H to a state difference as $H[{}^{(\alpha)}\mathbf{x}^f - \overline{\mathbf{x}^f}]$ will yield a different result than the operation $H[{}^{(\alpha)}\mathbf{x}^f] - H[\overline{\mathbf{x}^f}]$. Hence, equations (2.52) and (2.53) have to be reformulated with nonlinear operators H as

$$\tilde{\mathbf{P}}^f\mathbf{H}^T := \frac{1}{N-1}\Big(\mathbf{X}^f - \overline{\mathbf{X}^f}\Big)\Big(H[\mathbf{X}^f] - \overline{H[\mathbf{X}^f]}\Big)^T , \tag{2.75}$$

$$\mathbf{H}\tilde{\mathbf{P}}^f\mathbf{H}^T := \frac{1}{N-1}\Big(H[\mathbf{X}^f] - \overline{H[\mathbf{X}^f]}\Big)\Big(H[\mathbf{X}^f] - \overline{H[\mathbf{X}^f]}\Big)^T \tag{2.76}$$

where $H[\mathbf{X}^f]$ denotes the operation of H on all columns of $\mathbf{X}^f$. The notations on the left hand side of the equations have to be considered as symbolic, since no simple matrix-matrix operations are performed. Next to these equations, equation (2.50) for the influence amplitudes reads

$$\Big(\mathbf{H}\tilde{\mathbf{P}}^f\mathbf{H}^T + \mathbf{R}\Big)\,\mathbf{B} = \mathbf{Y}^o - H[\mathbf{X}^f] \tag{2.77}$$

Using the SEIK filter, the nonlinear measurement operator can also be applied. For this the term $\mathbf{HL}$ in equations (2.67) and (2.69) has to be replaced by $(H[\mathbf{X}^f])\mathbf{T}$. In addition equation (2.68) has to be written as

$$\mathbf{x}^a = \overline{\mathbf{x}^f} + \check{\mathbf{K}}\left(\mathbf{y}^o - \overline{H[\mathbf{x}^f]}\right) , \tag{2.78}$$

With these replacements the ensemble formulations used in the EnKF and SEIK algorithms do no more require the linearized operator $\mathbf{H}$. Despite this, these formulations comprise the problem that the analysis will not yield an optimal result of minimal variance since the analysis probability density will not be Gaussian.

2.5.3 State Augmentation to avoid Nonlinear Measurement Operators

To avoid the use of a nonlinear measurement operator, it has been proposed, see e.g. [1, 18, 4], to augment the state vector by the diagnostic variables. In this case, the measurement operator becomes trivially linear reducing the augmented state to the diagnostic variables.

For the state augmentation consider the state vector $\mathbf{x} \in \mathbb{R}^n$ and the observations $\mathbf{y}^o = H[\mathbf{x}] + \epsilon \in \mathbb{R}^n$. Now the augmented model state vector $\hat{\mathbf{x}} \in \mathbb{R}^{m+n}$ is defined by

$$\hat{\mathbf{x}} = \begin{pmatrix} \mathbf{x} \\ H[\mathbf{x}] \end{pmatrix} . \tag{2.79}$$

The ensemble matrix holding the augmented state vectors is then $\hat{\mathbf{X}} = \{{}^{(1)}\hat{\mathbf{x}}, \ldots, {}^{(N)}\hat{\mathbf{x}}\}$. Now, the measurement model is linear. It is given

$$\mathbf{y}^o = \hat{\mathbf{H}}\hat{\mathbf{x}}^t + \boldsymbol{\epsilon} \tag{2.80}$$

with the new linear measurement operator $\hat{\mathbf{H}} = (\mathbf{0}_{m\times n} \;\; \mathbf{1}_{m\times m})$.

We can rewrite the analysis equations (2.50) and (2.51) of the EnKF filter as

$$\mathbf{X}^a = \mathbf{X}^f + \hat{\mathbf{P}}^f\hat{\mathbf{H}}^T\mathbf{B} \tag{2.81}$$

where $\mathbf{B}$ is computed from

$$\left(\hat{\mathbf{H}}\hat{\mathbf{P}}^f\hat{\mathbf{H}}^T + \mathbf{R}\right)\mathbf{B} = \mathbf{Y}^o - \hat{\mathbf{H}}\hat{\mathbf{X}}^f \;. \tag{2.82}$$

In equation (2.81) we consider only the update of the first n elements in the state vectors. The augmented part is not changed by the update.

The representer matrix $\hat{\mathbf{P}}^f\hat{\mathbf{H}}^T$ and the matrix $\hat{\mathbf{H}}\hat{\mathbf{P}}^f\hat{\mathbf{H}}^T$ are given by

$$\hat{\mathbf{P}}^f\hat{\mathbf{H}}^T = \frac{1}{N-1}\left(\mathbf{X}^f - \overline{\mathbf{X}^f}\right)\left[\hat{\mathbf{H}}\left(\hat{\mathbf{X}}^f - \overline{\hat{\mathbf{X}}^f}\right)\right]^T \;, \tag{2.83}$$

$$\hat{\mathbf{H}}\hat{\mathbf{P}}^f\hat{\mathbf{H}}^T = \frac{1}{N-1}\hat{\mathbf{H}}\left(\hat{\mathbf{X}}^f - \overline{\hat{\mathbf{X}}^f}\right)\left[\hat{\mathbf{H}}\left(\hat{\mathbf{X}}^f - \overline{\hat{\mathbf{X}}^f}\right)\right]^T \;. \tag{2.84}$$

Using equations (2.81) to (2.84) the analysis update can be performed applying only the linear measurement operator $\hat{\mathbf{H}}$.

On the other hand, when the operation of $\hat{\mathbf{H}}$ in equations (2.82) to (2.84) is performed and the definition (2.79) of the augmented state is used it is

$$\left(\hat{\mathbf{H}}\hat{\mathbf{P}}^f\hat{\mathbf{H}}^T + \mathbf{R}\right)\mathbf{B} = \mathbf{Y}^o - H[\mathbf{X}^f] \tag{2.85}$$

and

$$\hat{\mathbf{P}}^f\hat{\mathbf{H}}^T := \frac{1}{N-1}\left(\mathbf{X}^f - \overline{\mathbf{X}^f}\right)\left[\left(H[\mathbf{X}^f] - \overline{H[\mathbf{X}^f]}\right)\right]^T \;, \tag{2.86}$$

$$\hat{\mathbf{H}}\hat{\mathbf{P}}^f\hat{\mathbf{H}}^T := \frac{1}{N-1}\left(H[\mathbf{X}^f] - \overline{H[\mathbf{X}^f]}\right)\left[\left(H[\mathbf{X}^f] - \overline{H[\mathbf{X}^f]}\right)\right]^T \;. \tag{2.87}$$

Equations (2.85) and (2.87) are identical to equations (2.75) and (2.77) formulated for the direct application of the nonlinear operator H discussed in section 2.5.2. Thus, the method of state augmentation is in fact equivalent to the direct application of the nonlinear measurement operator.

The logical fault in considering the method of state augmentation as the solution to cope with nonlinear measurement operators is that, despite the linear measurement operator, the distribution of the diagnostic variables $H[\mathbf{x}]$ will not be Gaussian. This is hidden in the formulation and likely to be overlooked. As the problems of state augmentation and direct application of H are the same, the latter method should be used in numerical applications. It does not produce computational overhead due to larger memory requirements for the state allocation.

2.6 Summary

Three different filter algorithms based on the Kalman filter have been motivated and discussed in the context of statistical estimation. These have been the EnKF, SEEK, and SEIK algorithms. These filter algorithms use a low-rank representation of the state covariance matrix and perform an analysis derived from the Extended Kalman filter (EKF). Due to this, we refer to these algorithms as Error Subspace Kalman Filters (ESKF). The ESKF algorithms have been related to the EKF. In addition, possible variations of the algorithms have been discussed.

The SEEK filter is a re-formulation of the EKF for a low-rank approximated state covariance matrix given decomposed form. This formulation reduces the computational costs to evaluate the forecast. In addition, the memory requirements are reduced by storing the covariance matrix in decomposed form. The EnKF filter applies a Monte Carlo method to sample and forecast the probability density function of the state estimate. In addition, the analysis computes the combination of two probability densities. These are the densities of the state estimate and of the observations. The analysis is performed by applying the analysis equations of the EKF to each ensemble state. The SEIK filter is an interpolated variant of the SEEK filter. Alternatively, it can be interpreted as an ensemble filter using a preconditioned ensemble. The SEIK algorithm uses an ensemble forecast as the EnKF filter. The analysis is computed analogous to the SEEK algorithm. The SEEK, EnKF, and SEIK algorithms will be compared more detailed in the next chapter.

Besides the ESKF algorithms, the problem of nonlinear measurement operators has been discussed. In this case, the filter estimate will be sub-optimal since the probability density of the analyzed state estimate will generally not be Gaussian. The ensemble based algorithms EnKF and SEIK show the advantage that they permit to apply the nonlinear operator directly. In contrast to this, the SEEK filter as well as the EKF require also the application of a linearized operator. It was also shown that including the diagnostic variables into the state vector, referred to as state augmentation, does only virtually solve the problem of nonlinear measurement operators. This method is equivalent to the direct application of the nonlinear operator.

Chapter 3

Comparison and Implementation of Filter Algorithms

3.1 Introduction

For the application of filter algorithms to geophysical modeling problems we are concerned with the search for filter algorithms for large-scale nonlinear systems. The three ESKF algorithms introduced in the previous chapter are compared under this aspect in section 3.2. Since all three filters owe the Extended Kalman Filter their similarity, the comparison focuses on the differences of the filters and consequences for their application to nonlinear systems. Further, relations to the error subspace are discussed. The EnKF and SEEK algorithms have also been compared by Brusdal et al. [7]. This work aimed at formulating the equations of the SEEK filter as similar as possible to the equations of the EnKF algorithm. Thus, the focus was rather on the similarity of the algorithms. Some of the results of the work by Brusdal et al. disagree with our comparison since the authors used also a formulation of the SEEK filter which differs from the formulation presented in section 2.4.1.

Besides the comparison of the algorithms, possible efficient implementations of the filters are presented in section 3.3. This includes a framework for filtering and the implementations of the analysis and resampling algorithms themselves. Finally, the computational complexity of the three filter algorithms is compared in section 3.4.

3.2 Comparison of SEEK, EnKF, and SEIK

All three algorithms have in common that they treat the covariance matrix $\mathbf{P}$ implicitly in some decomposed form. This avoids the requirement to compute $\mathbf{P}$ explicitly or to allocate storage for the whole covariance matrix. In addition, as all operations on $\mathbf{P}$ are symmetric, the covariance matrices remain symmetric throughout the computations.

3.2.1 Representation of Initial Error Subspaces

The initialization of the algorithms implies a different representation of their error subspaces representing the probability density $p(\mathbf{x}_0^t)$. The initial density $p(\mathbf{x}_0^t)$ is usually

assumed to be Gaussian or at least approximately Gaussian since the analysis phase of the filters also assumes a Gaussian density. Hence, $p(\mathbf{x}_0^t)$ is fully described by the state estimate $\mathbf{x}_0^a$ and the state covariance matrix $\mathbf{P}_0^a$. The Monte Carlo sampling used in the EnKF filter represents $p(\mathbf{x}_0^t)$ by a random ensemble of model state realizations. This approach permits, in general, to sample arbitrary probability densities. The sampling converges rather slow since the relative weights of the eigenvalues of $\mathbf{P}_0^a$, and hence the relative importances of the directions in the error subspace, are not taken into account. The statistics of the ensemble represent the error subspace. The SEEK and SEIK algorithms represent the error subspace at the state space point of the estimate $\mathbf{x}_0^a$ by the r major principal axes of the error ellipsoid described by the covariance matrix $\mathbf{P}_0^a$. This implies that the probability density is Gaussian or at least well described by $\mathbf{P}_0^a$. The SEEK filter treats the covariance matrix directly in it's decomposed form given by eigenvectors and a matrix of eigenvalues. The SEIK filter uses a statistical ensemble of minimum size, generated by minimum second-order exact sampling, whose ensemble statistics exactly represent the approximated $\mathbf{P}_0^a$. For SEEK and SEIK the convergence of the approximation with increasing r depends on the eigenvalue spectrum of $\mathbf{P}_0^a$. Typically, the sampling error in SEEK and SEIK will be much smaller then in the EnKF.

To exemplify the different sampling methods, figure 3.1 shows the sampling which represents the matrix

$$\mathbf{P}_t = \begin{pmatrix} 3.0 & 1.0 & 0.0 \\ 1.0 & 3.0 & 0.0 \\ 0.0 & 0.0 & 0.01 \end{pmatrix} . \tag{3.1}$$

$\mathbf{P}_t$ has the eigenvalues $\lambda_1 = 4, \lambda_2 = 2$, and $\lambda_3 = 0.01$. Thus, the smallest eigenvalue can be neglected to perform a low-rank approximation. The full matrix $\mathbf{P}_t$ can be represented by a probability ellipsoid in three dimensions while the low-rank approximation is represented by an ellipse. The sampling proposed for SEEK (upper left panel of figure 3.1) directly uses the eigenvectors of $\mathbf{P}_t$. In contrast, the RRSQRT algorithm [88], see also the remarks in section 2.4.1, uses modes which are scaled by the square root of the corresponding eigenvalue. Pure Monte Carlo sampling as used in the EnKF generates in this example an ensemble of much higher sampling errors. This is visible in the upper right panel for an ensemble size of $N = 100$. The second order exact sampling applied to initialize the SEIK filter is shown in the bottom panel. Here, three stochastic ensemble states represent exactly the low-rank approximated matrix $\mathbf{P}_t$.

The row-rank approximation used for second-order exact sampling assumes, that the major part of the model dynamics is represented by a limited number of modes or empirical orthogonal functions (EOFs). For realistic geophysical systems this requirement should be fulfilled, as has been shown, for example by Patil et al. [61] in the context of atmospheric dynamics.

Despite their different representations of the error subspace all three filters can be initialized from the same probability density or covariance matrix. For a consistent comparison of the filtering performance of different algorithms, it is even necessary to use the same initial conditions. Furthermore, the forecast and analysis equations of the EnKF and SEIK filters are in fact independent from the method the state ensembles

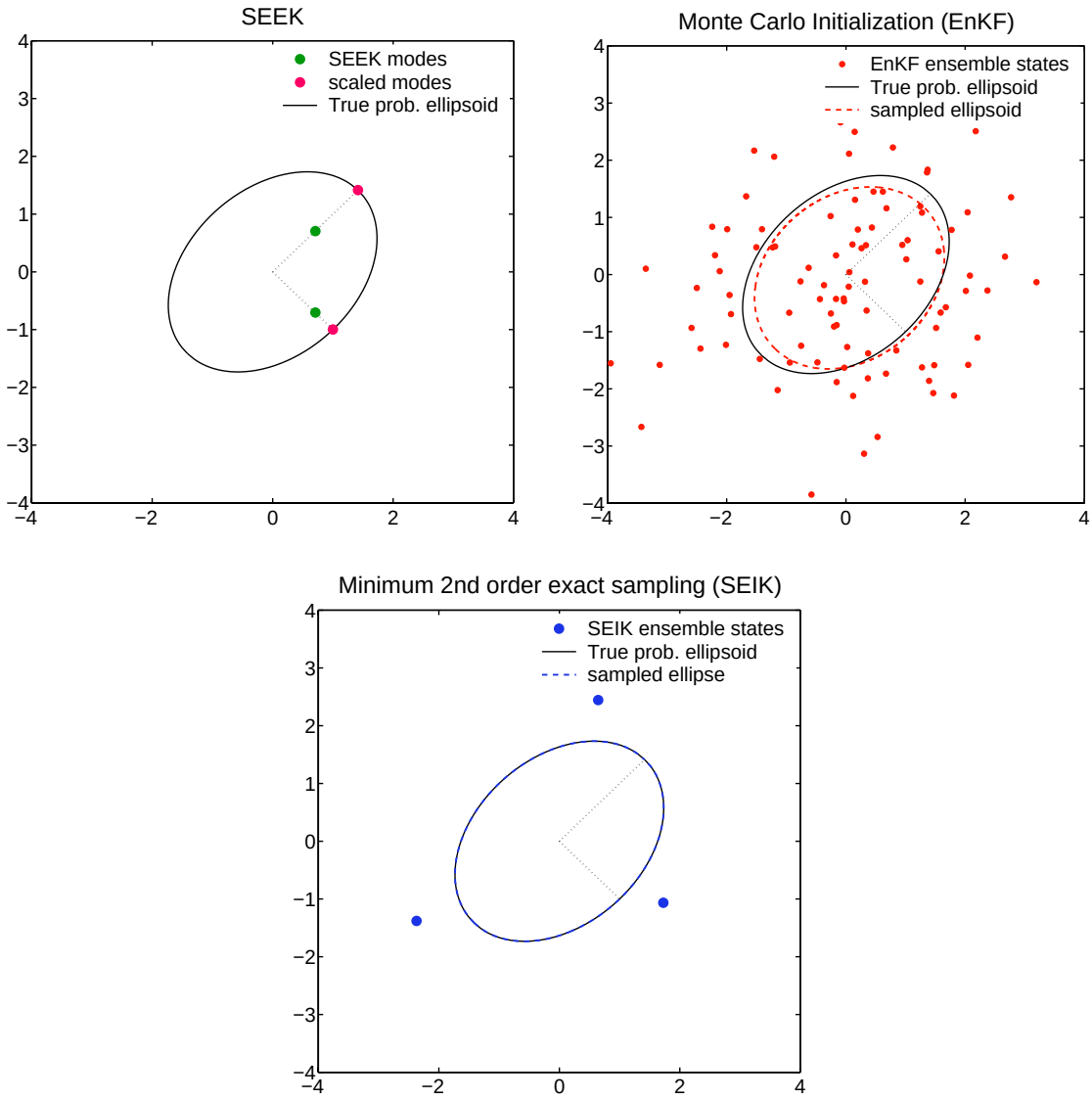


Figure 3.1: Sampling of a covariance matrix of rank 3 with SEEK (upper left), EnKF (upper right), and SEIK (bottom panel).

are generated. Thus, the initialization methods of Monte Carlo sampling and second-order exact sampling can be interchanged between EnKF and SEIK. Also the SEEK filter requires only the matrices $\mathbf{V}_0^a$ and $\mathbf{U}_0^a$, but it is independent from the method used to initialize these matrices. In general, the method to generate an initial state ensemble should hence be considered separately from the particular filter algorithm. It is still an open question which type of ensemble initialization will provide the best filter results in terms of the estimation error and the error in the estimated variance of the state estimate for a given ensemble size. The study of different initialization approaches is a topic of current research in meteorology, see e.g. [28, 29, 92].

3.2.2 Prediction of Error Subspaces

The forecast phase of the filter algorithms computes a prediction of the state estimate $\mathbf{x}_k^f$ and the error subspace at the next observation time t_k. The SEEK filter evolves the state estimate $\mathbf{x}_{k-\Delta k}^a$ with the nonlinear model to predict $\mathbf{x}_k^f$. To evolve the basis of the error subspace, the modes of $\mathbf{P}_{k-\Delta k}^a$ are evolved with the linearized model or a gradient approximation of it. In contrast to this, the EnKF and SEIK filters rely on nonlinear ensemble forecasting. Apart from the treatment of model errors, both algorithms evolve an ensemble of model states with the nonlinear dynamic model. The state estimate itself is not explicitly evolved as is done in the SEEK filter. The statistics of the forecasted ensemble represent the state estimate and forecast covariance matrix.

The explicit forecast of the state estimate by the SEEK filter only approximates the mean of the forecasted probability density. The ensemble forecast used in EnKF and SEIK accounts for higher order terms in the Taylor expansion, equation (2.20). Thus, these algorithms are expected to provide more realistic predictions of the error subspace compared with the SEEK filter. Concerning the forecast performed in SEEK, it can be dangerous to directly evolve the modes of $\mathbf{P}_{k-\Delta k}^a$, since this does not represent nonlinear interactions between different modes. Further, the increasingly finer scales of higher modes can lead to forecasts which do not provide meaningful directions of the error subspace.

3.2.3 Treatment of Model Errors

The SEEK and SEIK filters consider model errors by adding the model error covariance matrix $\mathbf{Q}$ to the forecasted state covariance matrix. The same is done in the EKF, except that the SEEK and SEIK algorithms neglect the parts of $\mathbf{Q}$ which are orthogonal to the error subspace. Alternatively, a simplified treatment is possible by applying the forgetting factor. This increases the variance in all directions of the error subspace by the same factor.

The EnKF applies a stochastic forcing during the ensemble forecast to account for model errors. Also it is possible to use a forgetting factor with the EnKF (See, for example, Hamill and Whitaker [30], where it is denoted as 'covariance inflation'). Since the SEIK filter also uses an ensemble forecast, it is possible to apply stochastic forcing in this algorithm, too.

In the context of a nonlinear system, the addition of $\mathbf{Q}$ at observation times is only an approximation. Over finite time the additive stochastic forcing in equation (2.1) will result in non-additive effects. Thus, applying stochastic forcing to the ensemble evolution will generally yield a more realistic representation of model errors than the addition of a matrix $\mathbf{Q}$. However, this requires the model errors to be known or, at least, to be well estimated. When the model errors are only poorly known, the forgetting factor provides a simple and numerically very efficient way to account for them. In addition, the forgetting factor can be applied to stabilize the filtering process by reducing the underestimation of the variances.

3.2.4 The Analysis Phase

The analysis phase of all three algorithms is based on the EKF analysis. Hence, only the first two statistical moments of the predicted probability density, the mean and covariance matrix, are taken into account. Thus, the analysis phase will provide only reasonable and approximately variance minimizing results if the predicted state probability density and the probability density of the observations are at least approximately Gaussian. For linear models the forecasted density is Gaussian if the initial density is Gaussian. For nonlinear systems the forecast density will contain a non-Gaussian part, but usually the state density will be close to Gaussian if a sufficient number of observations with Gaussian errors is taken into account as has been discussed by Brusdal et al. [7].

The increment for the analysis update is computed as a weighted average over vectors which belong to the error subspace $\tilde{\mathcal{E}}$. For SEEK these are the vectors in $\mathbf{V}$ and for SEIK the vectors in the matrix $\mathbf{L}$. In the case of EnKF the vectors are given by the difference $\mathbf{X}_k^f - \overline{\mathbf{X}_k^f}$ of the ensemble states to the ensemble mean. While SEEK and SEIK compute the weights for the analysis update in the error subspace, the EnKF computes the weights in the observation space. If a large amount of observational data is to be assimilated, i.e. if $m > N$, EnKF operates on matrices of larger dimension than SEEK and SEIK.

The analysis equations of SEEK are a re-formulation of the EKF update equations for a mode-decomposed covariance matrix $\hat{\mathbf{P}}_k^a = \mathbf{V}\mathbf{U}\mathbf{V}^T$. The forecast state estimate, given by the explicit evolution of $\mathbf{x}_{k-\Delta k}^f$, is updated using a Kalman gain computed from $\hat{\mathbf{P}}_k^a$ which itself is obtained by updating the matrix $\mathbf{U}_{k-\Delta k} \in \mathbb{R}^{r\times r}$. The analysis algorithms of EnKF and SEIK use the ensemble mean as forecast state estimate $\mathbf{x}_k^f$ and a covariance matrix $\tilde{\mathbf{P}}_k$ computed from the ensemble statistics. The SEIK filter updates the single state $\mathbf{x}_k^f$ and the eigenvalue matrix $\mathbf{U}_{k-\Delta k}$. The EnKF filter updates each ensemble member using for each update an observation vector from an ensemble of observations which needs to be generated. The analysis covariance matrix $\tilde{\mathbf{P}}_k^a$ is obtained implicitly by this ensemble analysis.

The requirement for an observation ensemble points to a possible drawback of the EnKF as, for finite ensembles, the observation ensemble will introduce additional sampling errors in the analyzed state ensemble. This is particularly pronounced if a large set, i.e. $m > N$, of independent observations is assimilated. In this case, the observation error covariance matrix $\mathbf{R}_k$ is diagonal having a rank of $m > N$. Thus, $\mathbf{R}_k$ cannot be well represented by an ensemble of size N.

For linear dynamic and measurement operators the predicted error subspace in the SEEK and SEIK algorithms will be identical if the same rank r is used and model errors are treated in the same way. Since also the analysis phases are equivalent both filters will yield identical results for linear systems. The filtering results of the EnKF will differ from that of the SEEK and SEIK filters even for linear dynamics and $N = r + 1$. This is due to the introduction of sampling noise by the Monte Carlo ensembles.

3.2.5 Resampling

Since the EnKF updates in the analysis phase the whole ensemble of model states the algorithm can proceed directly to the next ensemble forecast without the need of a resampling algorithm. In contrast to this, a new state ensemble representing $\check{\mathbf{P}}_k^a$ and $\mathbf{x}_k^a$ has to be generated when the SEIK filter is used. This can be done by a transformation of the forecast ensemble. Applying the SEEK filter, the forecasted modes of the covariance matrix can be used directly in the next forecast phase. In general, these are no more the basis vectors of the error subspace, since they are not orthonormal. A re-orthonormalization of the modes is recommendable and can be performed occasionally to stabilize the mode forecast. The choice whether an algorithm with or without re-initialization is used has no particular implications for the performance of the filter algorithms.

3.3 Implementation

For the implementation of the filter algorithms we aim at a modular structure which separates the routines of the model and filter parts of the program. In addition, the treatment of observations, e.g. the initialization of the observation vector or the measurement operator, should be dealt with separately from the model and the filter parts. Data should be exchanged between the three parts using interface routines.

Typically the filter has to be implemented with an existing model which is not designed for data assimilation purposes. Thus, the filter part should be attached to the model with minimal changes to the model source code and a clear interface structure. Here, we present an implementation of a serial filter environment which assumes that the time stepper part of the model is available as a subroutine. In chapter 8 we will present a framework for parallel data assimilation based on Kalman filter algorithms. It includes an application program interface, allows for efficient use of parallel computers, and does not require the model time stepper to be implemented as a subroutine. An interface structure between model and filter has also been discussed by Verlaan [87] in the context of the RRSQRT algorithm.

3.3.1 Main Structure of the Filter Algorithm

Besides the initialization, the filter algorithms consist of a forecast phase and an analysis phase. In addition, a resampling phase is performed by the SEEK and SEIK algorithms respectively for the modes or ensemble states.

To separate the filter part from the model we use a filter main routine which controls the ensemble forecast and subsequently calls subroutines performing the analysis and resampling phases of the algorithms. This filter main routine is called from the main program providing the fields for the filter initialization as subroutine arguments. These are either the initial state ensemble $\mathbf{X}_0$ (for EnKF and SEIK) or the initial state estimate $\mathbf{x}_0$ and matrices $\mathbf{U}_0$ and $\mathbf{V}_0$ (for SEEK). The initialization is performed in advance by some user written routine. The main routine for the SEIK filter is shown as algorithm 3.1 exemplifying the structure.

```
Subroutine SEIK_Main(n,N,X)
    int n  {state dimension, input}
    int N  {ensemble size, input}
    real X(n, N)  {state ensemble array, input}
    real x(n)  {state estimate}
    real Uinv(N − 1, N − 1)  {inverse of eigenvalue matrix}
    int i  {ensemble loop counter}
    int step  {time step counter}
    int m  {dimension of observation vector}
    real t_a  {physical time}

 1:   call User_Analysis_seik(0,n,N,X)  {call to user analysis routine}
 2:   loop
 3:     call Next_Observation(step, nsteps, t_a)
                  {get number of time steps, user supplied}
 4:     if nsteps = 0 then
 5:       exit loop
 6:     end if
 7:     for i=1 to N do
 8:       call Interface_Evolver(n,X(N),nsteps, t_a)
                    {forecast state vector, user supplied}
 9:     end for
10:     step ← step + nsteps
11:     call User_Analysis(−step,n,N,X)  {call to user supplied analysis routine}
12:     call SEIK_Analysis(step,n,N,x,Uinv,X)  {perform filter analysis phase}
13:     call SEIK_Resample(n,N,x,Uinv,X)  {perform ensemble resampling}
14:     call User_Analysis(step,n,N,X)  {call to user supplied analysis routine}
15:   end loop
```

Algorithm 3.1: Structure of the filter main subroutine for the SEIK algorithm. The arrays **x** and **Uinv** are required for the resampling computed in *SEIK_Resample*. They are initialized in the analysis routine *SEIK_Analysis*.

The calls to the subroutine *User_Analysis* in algorithm 3.1 provide the possibility to examine the assimilation progress during the execution. Here the user can analyze either the forecast or the analysis state ensemble. To distinguish both cases, the subroutine is called with the negative of the time step index *steps* in the forecast case. The routine permits, e.g., to compute ensemble means or variances estimated by the filter. In addition, the ensemble or analysis quantities can be written to files. For physical consistency it can be necessary to post-process the analysis states, for example to ensure mass conservation of a model ocean. This post-processing can be also performed in *User_Analysis* when called after the filter analysis phase.

In the forecast phase an ensemble of N model state vectors $\mathbf{X} = \{{}^{(1)}\mathbf{x}, \ldots, {}^{(N)}\mathbf{x}\}$ is evolved for *nsteps* time steps from the model time t_a to the time $t_b = t_a + nsteps \cdot \Delta t$ where Δt is the time step size. This requires to perform N model evolutions beginning

from the same model time t_a. The ensemble forecast is controlled by the filter, since the model does not need to consider filter details. The parameters *nsteps* and t_a are dependent on the data assimilation problem rather than on the model or the filter algorithm. Thus, they have to be provided by the user. For flexibility and to achieve a clear structure we implement the initialization of *nsteps* and t_a by a call to the user supplied subroutine *Next_Observation*. It has as input the current time step *step*. Outputs are *nsteps* and t_a.

Having obtained the values of *nsteps* and t_a, the forecast is performed in a loop over all ensemble vectors. Each of the vectors is handed over to the subroutine *Interface_Evolver* together with the stepping information. This interface routine initializes the state fields of the model from the state vector and calls the time stepper routine of the model. Finally the fields are written back into the state vector and the routine returns. Since *Interface_Evolver* is model dependent it has to be supplied by the user. The forecast phase requires that the N model evolutions are independent. Thus, any reused variables of the model have to be re-initialized.

Subsequent to the forecast phase, the analysis will be computed. In algorithm 3.1 this is performed in the subroutine *SEIK_Analysis*. We discuss the implementation of the analysis phases of the three filters in the following section. Finally, the ensemble will be resampled in the SEIK algorithm. The new ensemble is computed in the subroutine *SEIK_Resample*. The implementation of the resampling phases of SEIK and SEEK is described in section 3.3.3.

Subroutine SEEK_Main(n,r,$\mathbf{x}$,$\mathbf{Uinv}$,$\mathbf{V}$)
int r {rank, input}
real $\mathbf{x}(n)$ {state estimate, input}
real $\mathbf{Uinv}(r,r)$ {inverse of eigenvalue matrix, input}
real $\mathbf{V}(n,r)$ {mode matrix, input}
real ϵ {coefficient for gradient approximation}

⋮
1: **for** i=1 to r **do**
2: $\mathbf{V}(:,r) \leftarrow \mathbf{x} + \epsilon\mathbf{V}(:,r)$ {generate ensemble from modes}
3: **end for**
4: **for** i=1 to r **do**
5: call Interface_Evolver(n,$\mathbf{V}$(:,r),$nsteps, t_a$) {forecast ensemble vector}
6: **end for**
7: call Interface_Evolver(n,$\mathbf{x}$,$nsteps, t_a$) {forecast central state vector}
8: **for** i=1 to r **do**
9: $\mathbf{V}(:,r) \leftarrow \epsilon^{-1}(\mathbf{x} - \mathbf{V}(:,r)$ {generate forecast modes from ensemble}
10: **end for**
⋮

Algorithm 3.2: Structure of forecast part of the filter main subroutine for the SEEK algorithm

The structure of the main routine of the EnKF algorithm is analogous to that of the SEIK filter and thus not shown. The only functional difference is that the EnKF algorithm does not call a resampling routine. Further, the arrays **Uinv** and **x** are not required. For the SEEK algorithm the forecast part is different from the two other algorithms. In SEEK the state vector $\mathbf{x}$ and the mode matrix $\mathbf{V}$ are evolved. The structure of the forecast loop using a gradient approximation for the evolution of the modes stored in $\mathbf{V}$ is shown in algorithm 3.2.

3.3.2 The Analysis Phase

For the discussion of the implementation of the analysis phase we omit the time index from the equations. The analysis algorithms of the filter algorithms are shown in pseudo code as algorithms 3.3 to 3.4. Implemented are the analysis equations (2.28) to (2.30) of SEEK and (2.67) to (2.69) of SEIK. The EnKF analysis algorithm is implemented using the representer formulation according to equations (2.46) and (2.47). Further the ensemble representation of matrix $\mathbf{H}\tilde{\mathbf{P}}^f\mathbf{H}^T$ in equation (2.49) is used.

The analysis equations contain references to quantities which are dependent on the observations. The necessary observation-related operations in the source code for the filter analysis phase are:

- Query the dimension m of the observation vector (subroutine *Get_Dim_Obs*). The dimension m is required for dynamic allocation of arrays which are related to the observation space.
- Project a model state vector onto the observation space by applying the measurement operator $\mathbf{H}$ (subroutine *Measurement_Operator*).
- Initialize the observation vector y^o (subroutine *Measurement* for SEEK and SEIK). For EnKF an ensemble of observation vectors $\mathbf{Y}^o = \{^{(1)}\mathbf{y}^o, \ldots, ^{(N)}\mathbf{y}^o\}$ has to be generated according to the observation error covariance matrix $\mathbf{R}$. This is done in the subroutine *EnKF_Obs_Ensemble*.
- For SEEK and SEIK: Compute the product of the inverse of the observation error covariance matrix $\mathbf{R}$ with the matrix of modes projected on the observation space ($\mathbf{HV}$ for SEEK and $\mathbf{HX}$ for SEIK). This is performed in the subroutine *RinvA*.
- For EnKF: Add $\mathbf{R}$ to the state covariance matrix projected onto the observation space (subroutine *RplusA*).

These operations are implemented using subroutines which are provided by the user. This ensures modularity and keeps the analysis routines independent from the particular implementation of the measurement operator $\mathbf{H}$, the initialization of the observation vector y^o, and the implementation of the observation error covariance matrix $\mathbf{R}$.

This structure also permits, e.g., for the implementation of the product with $\mathbf{R}^{-1}$ or the addition of $\mathbf{R}$ in operational form, without explicit allocation of the matrix $\mathbf{R}$

or its inverse. As well the measurement operator can be implemented as an operation rather than a matrix multiplication. This implementation permits also the application nonlinear measurement operators which cannot be represented as a matrix. A further documentation of the observation-related subroutines is provided in appendix B.

In algorithm 3.3, the structure of the SEEK analysis routine with all calls to observation related subroutines is shown. The analysis routine of SEEK is the simplest of all three algorithms considered here.

Subroutine SEEK_Analysis($step$,n,r,$\mathbf{x}$,$\mathbf{Uinv}$,$\mathbf{V}$)
int $step$ {time step counter,input}
int n {state dimension, input}
int r {rank of covariance matrix, input}
real $\mathbf{x}(n)$ {state forecast, input/output}
real $\mathbf{Uinv}(r,r)$ {inverse eigenvalue matrix, input/output}
real $\mathbf{V}(n,r)$ {mode matrix, input/output}
real $\mathbf{T1}, \mathbf{T2}, \mathbf{t3}, \mathbf{t4}, \mathbf{d}, \mathbf{y}$ {local fields to be allocated}
int m {dimension of observation vector}
int i {ensemble loop counter}

1: call Get_Dim_Obs($step, m$) {get observation dimension, user supplied}
2: Allocate fields: $\mathbf{T1}(m,r), \mathbf{T2}(m,r), \mathbf{t3}(r), \mathbf{t4}(r), \mathbf{d}(m), \mathbf{y}(m)$

3: **for** i=1,r **do**
4: call Measurement_Operator($step, n, m, \mathbf{V}(:,i), \mathbf{T1}(:,i)$) {user supplied}
5: **end for**
6: call RinvA($step, m, r, \mathbf{T1}, \mathbf{T2}$) {user supplied}
7: $\mathbf{Uinv} \leftarrow \mathbf{Uinv} + \mathbf{T1}^T\mathbf{T2}$ {with BLAS routine DGEMM}

8: call Measurement_Operator($step, n, m, \mathbf{x}, \mathbf{d}$) {user supplied}
9: call Measurement($step, m, \mathbf{y}$) {user supplied}
10: $\mathbf{d} \leftarrow \mathbf{y} - \mathbf{d}$

11: $\mathbf{t3} \leftarrow \mathbf{T2}^T\mathbf{d}$ {with BLAS routine DGEMV}
12: solve $\mathbf{Uinv}\ \mathbf{t4} = \mathbf{t3}$ for $\mathbf{t4}$ {using LAPACK routine DGESV}
13: $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{V}\ \mathbf{t4}$ {update state estimate with BLAS routine DGEMV}
14: De-allocate local analysis fields

Algorithm 3.3: Structure of the filter analysis routine for the SEEK algorithm without handling of the model error covariance matrix. The subroutines called in the code are the observation-dependent operations described in section 3.3.2 and documented in appendix B. The matrices **T1**, **T2** and the vectors **t3**, **t4**, and **d** are temporary arrays. Other matrices and arrays appear which the same notation in equations (2.28) to (2.30).

Subroutine SEIK_Analysis($step$,n,N,$\mathbf{x}$,$\mathbf{Uinv}$,$\mathbf{X}$)
int $step$ {time step counter,input}
int n {state dimension, input}
int N {ensemble size, input}
real $\mathbf{x}(n)$ {state estimate, output}
real $\mathbf{Uinv}(r,r)$ {inverse eigenvalue matrix, output}
real $\mathbf{X}(n,N)$ {ensemble matrix, input/output}
real $\mathbf{G},\mathbf{d},\mathbf{y}$ {local fields to be allocated}
real $\mathbf{T1},\mathbf{T2},\mathbf{T3},\mathbf{t4},\mathbf{t5},\mathbf{t6}$ {local fields to be allocated}
int m {dimension of observation vector}
int i {ensemble loop counter}
int r {rank of covariance matrix, $r = N-1$}

1: call Get_Dim_Obs($step, m$) {get observation dimension, user supplied}
2: Allocate fields: $\mathbf{T1}(m,N), \mathbf{T2}(m,r), \mathbf{T3}(m,r), \mathbf{y}(m), \mathbf{t4}(r), \mathbf{t5}(r), \mathbf{t6}(N)$,
3: $\mathbf{G}(r,r), \mathbf{Uinv}(r,r), \mathbf{d}(m)$

4: **for** i=1,N **do**
5: call Measurement_Operator($step, n, m, \mathbf{X}(:,i), \mathbf{T1}(:,i)$) {user supplied}
6: **end for**
7: $\mathbf{T2} \leftarrow \mathbf{T1}\ \mathbf{T}$ {implemented with $\mathbf{T}$ as operator}
8: call RinvA($step, m, r, \mathbf{T2}, \mathbf{T3}$) {user supplied}
9: $\mathbf{G} \leftarrow N^{-1}(\mathbf{T}^T\ \mathbf{T})^{-1}$ {implemented as direct initialization}
10: $\mathbf{Uinv} \leftarrow \mathbf{G} + \mathbf{T2}^T\mathbf{T3}$ {with BLAS routine DGEMM}

11: $\mathbf{x} \leftarrow N^{-1}\sum_{i=1}^{N}\mathbf{X}(:,i)$ {get state estimate as ensemble mean state}
12: call Measurement_Operator($step, n, m, \mathbf{x}, \mathbf{d}$) {user supplied}
13: call Measurement($step, m, \mathbf{y}$) {user supplied}
14: $\mathbf{d} \leftarrow \mathbf{y} - \mathbf{d}$

15: $\mathbf{t4} \leftarrow \mathbf{T3}^T\mathbf{d}$ {with BLAS routine DGEMV}
16: solve $\mathbf{Uinv}\ \mathbf{t5} = \mathbf{t4}$ for $\mathbf{t5}$ {using LAPACK routine DGESV}
17: $\mathbf{t6} \leftarrow \mathbf{T}\ \mathbf{t5}$ {implemented with $\mathbf{T}$ as operator}
18: $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{X}\ \mathbf{t6}$ {update state estimate with BLAS routine DGEMV}
19: De-allocate local analysis fields

Algorithm 3.4: Structure of the filter analysis routine for the SEIK algorithm. The subroutines called in the code are the observation-dependent operations described in section 3.3.2 and documented in appendix B. The arrays $\mathbf{G}$ and $\mathbf{T2}$ are introduced for clarity. They do not need to be allocated since their contents are stored respectively in $\mathbf{Uinv}$ and $\mathbf{T1}$. The array $\mathbf{t5}$ is stored analogously in $\mathbf{t4}$.

The analysis routine of SEIK, shown as algorithm 3.4, is very similar to that of SEEK. It contains some additional operations like the initialization of the matrix $\mathbf{G}$ in line 9 and the computation of the ensemble mean in line 11. Also the matrix $\mathbf{T}$, defined by equation (2.62), has to be applied twice. For efficiency, the matrix $\mathbf{L} = \mathbf{XT}$ is not explicitly computed according to equation (2.64). Instead, $\mathbf{T}$ is applied in two different ways. First, the matrix $\mathbf{HL}$ is computed in lines 4 to 7 of algorithm 3.4. For this, the state ensemble is first projected onto the observation space yielding $\mathbf{HX}$. Subsequently, matrix $\mathbf{T}$ is applied as $(\mathbf{HX})\mathbf{T}$. To complete the computation of the analysis state, the equation

$$\mathbf{x}^a = \overline{\mathbf{x}^f} + \mathbf{XTa} \tag{3.2}$$

has to be evaluated with $\mathbf{a}$ given by

$$\mathbf{a} = \mathbf{U}\mathbf{L}^T\mathbf{H}^T\mathbf{R}^{-1}(\mathbf{y}^o - \mathbf{H}\overline{\mathbf{x}^f}) \ . \tag{3.3}$$

Here it is more efficient to act with $\mathbf{T}$ on the vector $\mathbf{a} \in \mathbb{R}^{(N-1)}$ instead on the ensemble matrix $\mathbf{X} \in \mathbb{R}^{n\times N}$. Since the structure of $\mathbf{T}$ is known, the product of some matrix or vector with $\mathbf{T}$ does not need to be computed as a full matrix-matrix product. The operation $(\mathbf{HX})\mathbf{T}$ involves the computation of the ensemble mean vector of $\mathbf{HX}$. This is then subtracted from the first r columns of $\mathbf{HX}$. The last column of this matrix is set to zero. Thus, the right-hand-side multiplication with $\mathbf{T}$ can be performed in place. It does only require the temporary allocation of a vector holding the ensemble mean. Further, only $2mN + m$ floating point operations are required for the application of $\mathbf{T}$ on $\mathbf{HX}$. The full matrix-matrix product would require mN^2 floating point operations. The operation $\mathbf{b} = \mathbf{Ta}$ involves the computation of the mean over the elements of $\mathbf{a}$. To obtain $\mathbf{b} \in \mathbb{R}^N$ the mean value is subtracted from each element of $\mathbf{a}$. The last entry in $\mathbf{b}$ is initialized by the negative value of the computed mean. The computation of $\mathbf{b}$ requires $2N$ floating point operations.

The analysis routine of EnKF is shown as algorithm 3.5. Using the representer formulation it is most efficient to perform the ensemble update in matrix form. That is, the residuals $\{\mathbf{d}^{(\alpha)}\}$ are stored in the columns of a matrix $\mathbf{D}$, then all influence amplitudes $\{\mathbf{b}^{(\alpha)}\}$ are computed at once as the matrix $\mathbf{B}$. Subsequently, all state vectors in the ensemble matrix $\mathbf{X}$ are updated at once. This procedure requires more computer memory, but it can be more efficiently optimized by compilers than a serial version executing a loop in which for each single residual vector a vector of influence amplitudes and finally a single updated ensemble state are computed. The second application of the measurement operator in line 14 is only shown to stress the similarity of the algorithms, but it is not required since the loop initializing the representer matrix in line 14 to 17 can be executed directly after the initialization of $\mathbf{T1}$ in lines 4 to 6.

Algorithm 3.5 shows the implementation of the analysis for large data sets when m is not significantly smaller than the ensemble size N. In this case, the matrix $\tilde{\mathbf{P}}^f\mathbf{H}^T \in \mathbb{R}^{n\times m}$, given by equation (2.48), is not explicitly computed. It is more efficient to compute the update of the ensemble states in equation (2.46) in the form

$$\mathbf{X}^a = \mathbf{X}^f + (\mathbf{X}^f - \overline{\mathbf{X}}^f)\mathbf{C} \tag{3.4}$$

Subroutine EnKF_Analysis($step$,n,N,$\mathbf{X}$)
int $step$ {time step counter,input}
int n {state dimension, input}
int N {ensemble size, input}
real $\mathbf{X}(n, N)$ {ensemble matrix, input/output}
real $\mathbf{D}, \mathbf{B}, \mathbf{x}, \mathbf{T1}, \mathbf{t2}, \mathbf{T3}, \mathbf{t4}, \mathbf{T5}, \mathbf{T6}$ {local fields to be allocated}
int m {dimension of observation vector}
int i {ensemble loop counter}

call Get_Dim_Obs($step, m$) {get observation dimension, user supplied}
Allocate fields: $\mathbf{T1}(m, N), \mathbf{t2}(m), \mathbf{T3}(m, m), \mathbf{t4}(m), \mathbf{T5}(n, N), \mathbf{T6}(N, N)$,
  $\mathbf{B}(m, N), \mathbf{D}(m, N), \mathbf{x}(n)$

for i=1,N **do**
  call Measurement_Operator($step, n, m, \mathbf{X}(:, i), \mathbf{T1}(:, i)$) {user supplied}
end for
$\mathbf{t2} \leftarrow N^{-1} \sum_{i=1}^{N} \mathbf{T1}(:, i)$ {get mean of ensemble projected on observation space}
for i=1,N **do**
  $\mathbf{T1}(:, i) \leftarrow \mathbf{T1}(:, i) - \mathbf{t2}$
end for
$\mathbf{T3} \leftarrow (N-1)^{-1}\ \mathbf{T1}\ \mathbf{T1}^T$ {with BLAS routine DGEMM}

call Enkf_Obs_Ensemble($step$,m,N,$\mathbf{D}$) {initialize ensemble of observations}
for i=1,N **do**
  call Measurement_Operator($step, n, m, \mathbf{X}(:, i), \mathbf{t4}$) {user supplied}
  $\mathbf{D}(:, i) \leftarrow \mathbf{D}(:, i) - \mathbf{t4}$ {initialize ensemble of residuals}
end for
call RplusA($step$,m,$\mathbf{T3}$) {add matrix $\mathbf{R}$ to $\mathbf{T3}$, user supplied}
solve $\mathbf{T3}\ \mathbf{B} = \mathbf{D}$ for $\mathbf{B}$ {using LAPACK routine DGESV}

$\mathbf{x} \leftarrow N^{-1} \sum_{i=1}^{N} \mathbf{X}(:, i)$ {get state estimate as ensemble mean state}
for i=1,N **do**
  $\mathbf{T5}(:, i) \leftarrow \mathbf{X}(:, i) - \mathbf{x}$
end for
$\mathbf{T6} \leftarrow \mathbf{T1}^T\ \mathbf{B}$ {with BLAS routine DGEMM}
$\mathbf{X} \leftarrow \mathbf{X} + (N-1)^{-1}\ \mathbf{T5}\ \mathbf{T6}$ {with BLAS routine DGEMM}
De-allocate local analysis fields

Algorithm 3.5: Structure of the filter analysis routine for the EnKF algorithm using the represented update variant for a non-singular matrix $\mathbf{T3}$. Shown is the variant which yields optimal performance if the dimension m of the observation vector is larger than half the ensemble size N. The subroutines called in the code are the observation-dependent operations described in section 3.3.2 and documented in appendix B. The arrays $\mathbf{B}$ and $\mathbf{t4}$ are only introduced for clarity. They do not need to be allocated since their contents can be stored respectively in $\mathbf{D}$ and $\mathbf{t2}$.

Subroutine EnKF_Analysis($step$,n,N,$\mathbf{X}$)
⋮

1: call Get_Dim_Obs($step, m$) {get observation dimension, user supplied}
2: Allocate fields: $\mathbf{T1}(m,N), \mathbf{t2}(m), \mathbf{T3}(m,m), \mathbf{t4}(m), \mathbf{T5}(n,N), \mathbf{T6}(n,m),$
3: $\mathbf{B}(m,N), \mathbf{D}(m,N), \mathbf{x}(n)$
⋮
23: $\mathbf{T6} \leftarrow \mathbf{T5}\ \mathbf{T1}^T$ {with BLAS routine DGEMM}
24: $\mathbf{X} \leftarrow \mathbf{X} + (N-1)^{-1}\ \mathbf{T6}\ \mathbf{B}$ {with BLAS routine DGEMM}
⋮

Algorithm 3.6: Variant of the filter analysis routine for the EnKF algorithm using the represented update variant for a non-singular matrix **T3**. This variant will yield better performance if there are significantly less observations then ensemble members. If $n \gg m$, this limit is at $2m < N$.

Subroutine SEEK_Reortho(n,r,$\mathbf{Uinv}$,$\mathbf{V}$)
int n {state dimension, input}
int r {rank of covariance matrix, input}
real $\mathbf{Uinv}(r,r)$ {inverse eigenvalue matrix, input/output}
real $\mathbf{V}(n,r)$ {mode matrix, input/output}
real $\mathbf{T1}, \mathbf{T2}, \mathbf{T3}, \mathbf{T4}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \mathbf{L}, \mathbf{U}$ {local fields to be allocated}

1: Allocate fields: $\mathbf{T1}(r,r), \mathbf{T2}(r,r), \mathbf{T3}(r,r), \mathbf{T4}(r,r),$
2: $\mathbf{A}(r,r), \mathbf{B}(r,r), \mathbf{C}(r,r), \mathbf{D}(r,r), \mathbf{L}(n,r), \mathbf{U}(r,r)$

3: $\mathbf{U} \leftarrow \mathbf{Uinv}^{-1}$ {inversion using LAPACK routine DGESV}
4: Cholesky decomposition: $\mathbf{U} = \mathbf{A}\mathbf{A}^T$ {using LAPACK routine DPOTRF}
5: $\mathbf{T1} \leftarrow \mathbf{V}^T\ \mathbf{V}$ {with BLAS routine DGEMM}
6: $\mathbf{T2} \leftarrow \mathbf{T1}\ \mathbf{A}$ {with BLAS routine DGEMM}
7: $\mathbf{B} \leftarrow \mathbf{A}^T\ \mathbf{T2}$ {with BLAS routine DGEMM}

8: SVD: $\mathbf{B} = \mathbf{C}\ \mathbf{D}\ \mathbf{C}^T$ {using LAPACK routine DSYEV}
9: $\mathbf{T3} \leftarrow \mathbf{C}\ \mathbf{D}^{-1/2}$
10: $\mathbf{T4} \leftarrow \mathbf{A}\ \mathbf{T3}$ {with BLAS routine DGEMM}
11: $\mathbf{L} \leftarrow \mathbf{V}$
12: $\mathbf{V} \leftarrow \mathbf{L}\ \mathbf{T4}$ {with BLAS routine DGEMM}
13: $\mathbf{Uinv} \leftarrow \mathbf{D}^{-1}$
14: De-allocate local analysis fields

Algorithm 3.7: Structure of the re-orthonormalization routine for the SEEK algorithm. The matrix **D** holding the singular values of **B** is introduced here for clarity. In the program it is allocated as a vector holding the eigenvalues of **B**. The matrices **A**, **T1**, **C**, **T3**, and **T4** are not allocated in the program. Their information is stored in other arrays.

Subroutine SEIK_Resample(n,N,$\mathbf{x}$,$\mathbf{Uinv}$,$\mathbf{X}$)
int n {state dimension, input}
int N {ensemble size, input}
real $\mathbf{x}(n)$ {state analysis vector, input}
real $\mathbf{Uinv}(r, r)$ {inverse eigenvalue matrix, input}
real $\mathbf{X}(n, N)$ {ensemble matrix, input/output}
real $\mathbf{T1}, \mathbf{T2}, \mathbf{T3}, \mathbf{\Omega}^T, \mathbf{C}$ {local fields to be allocated}
int r {rank of covariance matrix, $r = N - 1$}

1: Allocate local analysis fields: $\mathbf{T1}(r, N), \mathbf{T2}(N, N), \mathbf{T3}(n, N), \mathbf{\Omega}^T(r, N), \mathbf{C}(r, r)$

2: Cholesky decomposition: $\mathbf{Uinv} = \mathbf{C}\ \mathbf{C}^T$ {using LAPACK routine DPOTRF}
3: initialize $\mathbf{\Omega}^T$ {implemented as a subroutine}
4: solve $\mathbf{C}^T\mathbf{T1} = \mathbf{\Omega}^T$ for $\mathbf{T1}$ {using LAPACK routine DTRTRS}
5: $\mathbf{T2} \leftarrow \mathbf{T}\ \mathbf{T1}$ {implemented with $\mathbf{T}$ as operator}
6: **for** i=1,N **do**
7: $\mathbf{T3}(:, i) \leftarrow \mathbf{X}(:, i)$
8: $\mathbf{X}(:, i) \leftarrow \mathbf{x}$
9: **end for**
10: $\mathbf{X} \leftarrow \mathbf{X} + N^{1/2}\ \mathbf{T3}\ \mathbf{T2}$ {with BLAS routine DGEMM}
11: De-allocate local analysis fields

Algorithm 3.8: Structure of the re-orthonormalization routine for the SEEK algorithm. The matrices $\mathbf{C}$ and $\mathbf{T1}$ are introduced here for clarity. In the program they are not allocated as their information is stored respectively in $\mathbf{Uinv}$ and $\mathbf{\Omega}^T$.

Subroutine SEEK_Reortho_Block(n,r,$\mathbf{Uinv}$,$\mathbf{V}$)
⋮
int $maxblksize$ {Maximum size for blocking}
int $blklower, blkupper$ {Counters for blocking}

1: Allocate fields: $\ldots, \mathbf{L}_b(blkmax, r)$
⋮
11: **for** $i = 1, n, maxblksize$ **do**
12: $blkupper \leftarrow \min(blklower + maxblksize - 1, n)$
13: $\mathbf{L}_b(1 : blkupper - blklower + 1, :) \leftarrow \mathbf{V}(blklower : blkupper, :)$
14: $\mathbf{V}(blklower : blkupper, :) \leftarrow \mathbf{L}_b(1 : blkupper - blklower + 1, :)\ \mathbf{T4}$
15: **end for**
⋮

Algorithm 3.9: Block formulation for the part of the re-orthonormalization routine of SEEK which initializes the new covariance modes. The block formulation replaces lines 11 and 12 of algorithm 3.7. The lower index b denotes that only a block of size $maxblksize \times r$ of the matrix $\mathbf{L}$ is allocated.

with

$$\mathbf{C} = \left[(N-1)^{-1} (\mathbf{H}\mathbf{X}^f - \mathbf{H}\overline{\mathbf{X}}^f)^T \right] \mathbf{B} \tag{3.5}$$

where $\overline{\mathbf{X}}^f \in \mathbb{R}^{n \times N}$ denotes the matrix holding the ensemble mean state $\overline{\mathbf{x}^f}$ in all columns. This update requires $(m+n)N^2$ operations, without the computation of the term in brackets in equation (3.5).

The alternative algorithm for small m is shown in algorithm 3.6. Here the matrix $\tilde{\mathbf{P}}^f \mathbf{H}^T$ is explicitly computed. Thus, nmN floating point operations are performed for equivalent computations to equations (3.4) and (3.5). If $n \gg m$, this alternative variant performs less floating point operations than the variant shown above for $2m < N$.

3.3.3 The Resampling Phase

The resampling phases of SEEK and SEIK are independent from model or observations. The implementation of the resampling algorithms is shown as algorithm 3.7 for the SEEK and 3.8 for the SEIK algorithm.

For SEEK the algorithm to re-orthonormalize the modes of the covariance matrix is implemented by first computing the product $\mathbf{V}^T\mathbf{V}$. This is a rather costly operation requiring nr^2 operations. The other products to complete the computation of $\mathbf{B}$ are only $\mathcal{O}(r^3)$. The resampling of the ensemble in SEIK (equation 2.71) involves again the matrix $\mathbf{L}$. As in the analysis algorithm, we do not compute this matrix explicitly. Instead, matrix $\mathbf{T}$ is applied from the left to the matrix $(\mathbf{\Omega}\mathbf{C}^{-1})^T \in \mathbb{R}^{(N-1)\times N}$. This operation is analogous to the operation $\mathbf{Ta}$ which was discussed for the analysis algorithm of SEIK. Since the application of $\mathbf{T}$ from the left acts on columns, the operation in the resampling corresponds to the application to N vectors. Thus, the application of $\mathbf{T}$ to a matrix is the generalization of the application to a vector.

3.3.4 Optimizations for Efficiency

The analysis and resampling phases contain several matrix-matrix and matrix-vector products. The sequences chosen for the computation of the products minimizes the size of the arrays to be allocated. For efficiency we implement the products using the highly optimized BLAS library routines. Other operations, like the Cholesky factorization in the resampling phase of SEIK, the eigenvalue decompositions, or the inversion of $\mathbf{U}^{-1}$ in the analysis phases of SEEK and SEIK are implemented using LAPACK library routines. The use of library functions is documented in the annotations in the algorithms 3.3 to 3.8.

All three analysis algorithms and both resampling algorithms allow for a block formulation of the final matrix-matrix product updating the ensemble or mode matrix. In some situations this can reduce the memory requirements of the algorithms and may lead to a better performance of the algorithms (if the BLAS routine itself does not use a blocking internally). In the context of the EnKF a block formulation has been discussed by Evensen [18]. To exemplify the block formulation we consider the resampling algorithm of SEEK. The variant without blocking is shown as algorithm 3.7

while the variant with blocking is displayed as algorithm 3.9. For the block algorithm a loop is constructed running from 1 to n with a step size of the chosen blocking size *maxblksize*. Within the loop, matrix $\mathbf{L}$ is allocated as a matrix $\mathbf{L}_b$ with only *maxblksize* rows. The loop counter determines which rows of $\mathbf{V}$ are updated in a single cycle. In each loop cycle only the corresponding rows of $\mathbf{L}$ are initialized in $\mathbf{L}_b$ and used to update the selected rows of $\mathbf{V}$. With the block formulation the required memory allocation for $\mathbf{L}$ can be significantly reduced from $n \times r$ to $maxblksize \times r$, where $maxblksize \approx 100, \ldots, 500$. In addition, the performance of the algorithm may be higher with the block formulation, since the smaller matrices may better fit into the caches of the processor. This would reduce costly transfers between the caches and the main memory of the computer.

3.4 Computational Complexity of the Algorithms

In most realistic filtering applications the major amount of computing time is spent for the model evolution. This time is proportional to the size of the ensemble to be evolved. It is equal for all three algorithms if $r+1 = N$ where r is the rank of the approximated covariance matrix in SEEK and SEIK and N is the ensemble size in EnKF. For efficient data assimilation it is thus of highest interest to find the algorithm which yields the best filtering performance, in terms of estimation error reduction, with the smallest ensemble size. The forecast phase consists of N independent model evaluations. This is also true for the SEEK filter if a gradient approximation of the linearized model is used. Distributing the model evaluations over multiple processors would permit to compute several model forecasts concurrently. Thus, the independence of the model forecasts can be utilized by parallelization. We will examine this possibility in detail in part 2 of this work.

The computation time spent in the analysis and resampling phases can also be non-negligible, especially if observations are frequently available. The three filter algorithms can show significant differences in these phases. Below we assume $n \gg m > N$. This situation occurs if we have a large scale model. Also m can be significantly larger than N, e.g., if data from satellite altimetry is used. Under this assumptions operations on arrays involving the dimension n are most expensive followed by operations on arrays involving the dimension m.

Table 3.1 shows the scaling of the computational complexity for the three filter algorithms. Since we are only interested in the scaling, we neglect in the table the difference between r and N. We use N if some operation is proportional to the ensemble size of the rank of the covariance matrix.

Without the explicit treatment of the model error covariance matrix $\mathbf{Q}$ the SEEK filter is the most efficient algorithm. All operations which depend on the state dimension n scale linear with n. These operations occur in the update of the state estimate in line 13 of algorithm 3.3. The matrix of weights for the state update is computed in the error space. Thus, the complexity of several operations depends on N. Most costly is the solver step in line 12 which scales with $\mathcal{O}(N^3)$. The product $\mathbf{R}^{-1}\mathbf{HV}$, which

Table 3.1: Overview of the scaling of the computational complexity of the filter algorithms. The scaling numbers only show the dependence of the three dimensions but no constant factors. The first column shows the number of the corresponding equation. The second column displays the corresponding rows of the algorithm which is named above each list. The scaling numbers neglect the difference between the ensemble size N and the rank r. Thus, the complexity is given in terms of N also for the SEEK filters.

SEEK analysis, algorithm 3.3

equation	lines	$\mathcal{O}$(operations)	comment
2.28	3-4	$m^2N + mN^2 + m + N \cdot h$	update $\mathbf{U}^{-1}$
2.29/2.30	8-10	$m + h$	initialize residual $\mathbf{d}$
2.29/2.30	11-13	$nN + n + mN + N^3 + N^2$	update state estimate $\mathbf{x}$
2.27		$n^2N + nN^2 + N^3$	compute $\hat{\mathbf{Q}}_k$

SEEK re-orthonormalization, algorithm 3.7

2.31	3-7	$nN^2 + N^3$	compute $\mathbf{B}$
2.32	8-13	$nN^2 + nN + N^3 + N^2$	compute $\hat{\mathbf{V}}$ and $\hat{\mathbf{U}}^{-1}$

SEIK analysis, algorithm 3.4

2.67	4-10	$m^2N + mN^2 + mN + N^2 + N \cdot h$	compute $\mathbf{U}^{-1}$
2.68/2.69	11-14	$mN + h$	initialize residual $\mathbf{d}$
2.68/2.69	15-18	$nN + n + mN + N^3 + N^2 + N$	update state estimate $\mathbf{x}$
2.27		$n^2N + nN^2 + N^3$	compute $\check{\mathbf{Q}}_k$

SEIK resampling, algorithm 3.8

2.71	1-5	$N^3 + N^2 + N$	compute $(\mathbf{C}^{-1}\mathbf{\Omega})^T$
2.71	6-10	$nN^2 + nN$	update ensemble $\mathbf{X}$

EnKF analysis, algorithm 3.5

2.49	4-11	$m^2N + mN + N \cdot h$	compute $\mathbf{H}\tilde{\mathbf{P}}^f\mathbf{H}^T$
2.47	12	$m^3 + m^2N + mN$	observation ensemble $\mathbf{Y}$
2.47	13-18	$m^3 + m^2N + mN$	representer amplitudes $\mathbf{B}$
3.4/3.5	19-24	$nN^2 + nN + mN^2$	update ensemble $\mathbf{X}$

is required in the update of $\mathbf{U}^{-1}$ in equation (2.28), is the only operation which can be proportional to $\mathcal{O}(m^2N)$. The full cost will only occur if different measurements are correlated. If the measurements are independent, the observation error covariance matrix $\mathbf{R}$ is diagonal. In this case, the products will scale with $\mathcal{O}(mN)$. Since the product is implemented as a subroutine, it can always be implemented in the optimal way depending on the structure of $\mathbf{R}^{-1}$.

The re-orthonormalization of the SEEK filter requires extensive operations on the matrix $\mathbf{V}$ which holds the modes of the covariance matrix. The complexity of the

computation of the product $\mathbf{V}^T\mathbf{V}$ (line 5 of algorithm 3.7) and the initialization of the new orthonormal modes in line 12 scales proportional to $\mathcal{O}(nN^2)$. Since it is only occasionally required to compute the re-orthonormalization, this operation will not affect the overall numerical efficiency of the SEEK filter.

The numerical complexity of the analysis phase of the SEIK filter is very similar to that of the SEEK algorithm. The computation of the ensemble mean state in line 11 of algorithm 3.4 will produce some overhead in comparison to the SEEK algorithm. Its complexity scales with $\mathcal{O}(nN + n)$. Other additional operations in comparison to the SEEK filter are applications of the matrix $\mathbf{T}$. As has been discussed above, these operations require $2mN + m + 2N$ floating point operations. Finally, the initialization of the matrix $\mathbf{G}$ is required. This will require N^2 operations, since it can be performed directly.

The resampling phase of SEIK is significantly faster than that of SEEK, since no diagonalization of $\check{\mathbf{P}}^a$ is performed. Hence, operations on matrices involving the state dimension n only occur in the ensemble update in lines 6 to 10 of algorithm 3.8. The complexity of these operations scale with $\mathcal{O}(nN^2 + nN)$. For rather large ensembles also the Cholesky decomposition in line 2 and the solver step in line 4 can be significant. The complexities of both operations scale with $\mathcal{O}(N^3)$. The cost of the initialization of the matrix $\mathbf{\Omega}$ can be neglected. For each resampling, the same matrix $\mathbf{\Omega}$ can be used in equation (2.71). Thus, $\mathbf{\Omega}$ can be stored.

The computational complexity of the SEEK and SEIK algorithms will increase strongly if the model error covariance matrix $\mathbf{Q}$ is taken into account. This is due to the amount of floating point operations required for the projection of $\mathbf{Q}$ onto the error space (cf. equation (2.27)). This projection requires $n^2N + 2nN^2 + 3N^3$ operations if $\mathbf{Q}$ has full rank. Due to the part scaling with $\mathcal{O}(n^2N)$, it is unfeasible to apply this projection. The amount of operations is significantly smaller if $\mathbf{Q}$ has a low rank of $k \ll n$ and is stored in square root form $\mathbf{Q} = \mathbf{A}\mathbf{A}^T$ with $\mathbf{A} \in \mathbb{R}^{n\times k}$. In this case, the projection requires $nN^2 + nkN + N^2k + 2N^3$ floating point operations. Thus, the complexity of the projection is comparable to the complexity of the resampling phases of SEEK and SEIK if the low-rank formulation for $\mathbf{Q}$ is used. However, also the low-rank formulation of the projection requires a very high amount of floating point operations. If the model errors are only poorly known it would probably be to expensive in terms of computation time to use this projection. Alternatively the forgetting factor could be used. The application of the forgetting factor requires N^2 floating point operations. In SEIK it is also possible to apply model errors as a stochastic forcing during the forecast phase. If this forcing is applied at every time step to each element of all ensemble states, the complexity of this technique scales with $\mathcal{O}(nN \cdot nsteps)$ for each time step.

The EnKF algorithm appears appealing as it does not require an explicit resampling of the ensemble. The ensemble states are updated during the analysis phase of the filter. The complexity of the ensemble update in line 24 of algorithm 3.5 scales with $\mathcal{O}(nN^2+nN)$. Hence, this operation is equivalent to the ensemble update in SEIK or the initialization of new modes in SEEK. In fact, the computation of new modes or ensemble states amounts for all three filters to the calculation of weighted averages of

the prior ensembles or modes. Since the EnKF uses the representer formulation which operates in the observation space, all other operations in the analysis algorithm are dependent on m. The complexity of the solver step for the representer amplitudes in line 18 of algorithm 3.5 scales with $\mathcal{O}(m^3 + m^2 N)$. Thus, this operation will be very costly if large observational data sets are assimilated. Costly will be also the computation of the matrix $\mathbf{H}\tilde{\mathbf{P}}^f\mathbf{H}^T$. The complexity of this operation is proportional to $\mathcal{O}(m^2 N)$. Another costly operation can be the generation of an ensemble of observations. This operation has to be supplied as a subroutine by the user of the filter. We use an implementation which applies a transformation of independent random numbers. It is described in detail in section 4.2. The transformation requires the eigenvalue decomposition of the covariance matrix $\mathbf{R}$ which scales with $\mathcal{O}(m^3)$. The complexity of the subsequent initialization of the ensemble vectors is proportional to $\mathcal{O}(m^2 N)$. Hence, the generation of the observation ensemble is of comparable complexity to the solver step for the representer amplitudes. Overall, the EnKF analysis requires more floating point operations than the SEEK and SEIK filters. This is caused by the representer formulation used in the EnKF algorithm. Due to this, the EnKF algorithm operates on the observation space rather than the error subspace which is directly taken into account by the SEEK and SEIK filters.

To optimize the performance of the EnKF and its ability to handle very large observational data sets, Houtekamer and Mitchell [36] discussed the use of an iterated analysis update. In this case, the observations are subdivided into batches of independent observations. Each iteration uses one batch of observations to update the ensemble states. Hence, the effective dimension of the observation vector is reduced. Since the EnKF contains several operations which scale with $\mathcal{O}(m^3)$ or $\mathcal{O}(m^2)$, this technique diminishes the complexity of the algorithm. In addition, the memory requirements are reduced. The iterative analysis update can also be applied with the SEEK and SEIK filters. In contrast to the EnKF algorithm, most operations in the analysis algorithms of SEEK and SEIK are proportional to $\mathcal{O}(m)$. Only the complexity of the matrix-matrix product implemented in the subroutine *RinvA* will scale with $\mathcal{O}(m^2)$ if $\mathbf{R}^{-1}$ is not diagonal. Hence, no particular performance gain can be expected for SEEK and SEIK when using batches of observations. The memory requirements are, however, reduced also for these filters.

Recently, Evensen [18] proposed an efficient analysis scheme for the EnKF which is based on a factorization of the term in parentheses in the Kalman gain equation (2.42). This relies on an ensemble representation of the observation error covariance matrix $\mathbf{R}$ and requires that the state and observation ensembles are independent. As has been discussed in the remarks on the EnKF, this scheme can lead to a further degradation of the filter quality. With this newer analysis scheme the complexity of operations which scale with m^3 or m^2 is reduced to be proportional to m. An exception from this is the generation of the observation ensemble which remains unchanged. Thus, apart from the generation of the observation ensemble, the complexity of the newly proposed EnKF analysis scheme will be similar to the complexities of SEEK and SEIK. However, the generation of the observation ensemble will remain costly.

3.5 Summary

The three error subspace Kalman filter algorithms introduced in chapter 2 have been compared. The comparison focused on the capabilities of the filter algorithms for data assimilation with large-scale nonlinear models. It became evident that the EnKF and SEIK filters are comparable as ensemble methods. They use, however, different initialization schemes for the ensembles. In addition, the analysis phase of the EnKF algorithm has a higher computational complexity if the dimension of the observation vector is larger than the ensemble size. This is due to the fact that the EnKF algorithm operates on the observation space rather than on the error subspace spanned by the ensemble states. The EnKF analysis also introduces noise into the state ensemble caused by the requirement of an ensemble of observation vectors. For finite ensembles, the observation ensemble will not exactly represent the observation error covariance matrix. The SEEK filter is initialized similarly to the SEIK algorithm. Also the analysis phases of both filters are rather similar. However, the SEEK filter applies a linearized forecast of the covariance modes which is distinct from the ensemble forecast used in the SEIK algorithm. Due to this, the error subspace predicted by the SEEK filter can be strongly distinct from that predicted by the SEIK filter.

It has been discussed, that the initialization of the filter algorithms should be considered separately from the analysis and resampling phases. In particular, the SEIK and the EnKF algorithm are independent from the method which is used to generate the state ensemble. Thus, also the EnKF algorithm can be initialized with a sampling scheme which yields a better representation of the state covariance matrix than pure Monte Carlo sampling.

The discussion of the implementation of the ESKF algorithms showed that the filter algorithms are relatively easy to implement since mostly algebraic operations are performed. The EnKF has the plainest structure but also the SEIK filter, using the most advanced mathematical formulation of the filters studied here, can be implemented with a few hundred lines of source code. For the implementation, the structure of a serial filtering framework was introduced. The framework is based on a clear separation of the model, the filter, and the observational part of the data assimilation problem. Main routines of the filter algorithms were implemented to control the phases of the filters. The forecast phase is performed by a loop over all ensemble states or modes. Subsequently the analysis and resampling routines of the filter algorithms are called. This structure will be extended to a filtering framework for parallel data assimilation with ESKF algorithms in chapter 8.

Chapter 4

Filtering Performance

4.1 Introduction

The previous chapters showed that the EnKF and SEIK filters both use nonlinear ensemble forecasting to predict error statistics. Due to the necessity of an ensemble of observations vectors in its analysis phase, the EnKF is likely to yield less realistic state and covariance estimates compared with the SEIK filter. This is due to noise inserted into the ensemble states by the observation ensemble. The SEEK algorithm re-formulates and approximates the Extended Kalman filter. This first order extension of the classical (linear) Kalman filter is expected to show limited abilities to handle nonlinearity.

Experimental studies of data assimilation with different filter algorithms showed that quite different ensemble sizes are required to obtain comparable results. Heemink et al. [31] reported that the RRSQRT filter yielded comparable estimation errors to the EnKF for about half the number of model evaluations in a study using a 2D advection diffusion equation. A comparison between SEEK and EnKF with an ocean general circulation model [7] used 8 model state evaluations for the SEEK filter and an ensemble size of 150 for the EnKF. With these numbers both filters obtained qualitatively comparable estimation errors. This result is, however, difficult to interpret since both filters where applied to slightly different model configurations and used different initial conditions for the filters.

In this chapter identical twin experiments are performed to assess the behavior of the SEEK , EnKF and SEIK algorithms when applied to a nonlinear oceanographic test model of moderate size. The experiments utilize shallow water equations with nonlinear evolution and synthetic observations of the sea surface height. Identical conditions for the algorithms are used. This permits a direct and consistent comparison of the filtering performances for various ensemble sizes. The experiments are evaluated by studying the filtering performance in terms of the root mean square (rms) estimation error for a variety of ensemble sizes. In addition, it is studied how the distinct representations of the covariance matrix and the different analysis schemes of the filter algorithms yield different filtering performances. This is done by a statistical examination of the quality of the sampled state covariance matrices, and hence the error subspaces represented by the filter algorithms.

In section 4.2 the configuration of the data assimilation experiments is described. Section 4.3 presents and discusses the results of the data assimilation experiments in terms of the estimation errors. Subsequently, the statistical examination of the quality of the sampled state covariance matrices is presented in section 4.4. Here additional quantities for the examination are defined and subsequently discussed.

4.2 Experimental Configurations

To assess the filtering abilities of the different filter algorithms identical twin experiments are performed with a toy model using the nonlinear shallow water equations, see e.g. [62],[1]

$$\partial_t \vec{u} + (\vec{u}\,\nabla)\vec{u} + \vec{f} \times \vec{u} + g\nabla h = 0 \tag{4.1}$$

$$\partial_t h + \nabla \cdot ((H_0 + h)\vec{u}) = 0 \tag{4.2}$$

where $\vec{u}(\vec{r},t) = (u(\vec{r},t), v(\vec{r},t))$ is the velocity field and $h(\vec{r},t)$ is the field of the sea surface elevation ($\vec{r} = (x,y)$ is the 2-dimensional location vector). $H_0(\vec{r},t)$ is the sea depth and g is the gravitational acceleration. Further, $\vec{f} = 2\Omega \sin\theta\ \vec{k}$, where Ω is the angular velocity of the Earth, θ is the latitude, and $\vec{k}$ is the vertical unit vector.

The shallow water equations are discretized in potential enstrophy conserving form according to Sadourny [71] with the extension to include the Coriolis term. The model domain is chosen as a box measuring 950 km per side with a flat bottom at 1000 meters depth. Periodic boundary conditions are applied in zonal and meridional directions. The Coriolis parameter $2\Omega \sin\theta$ is constant over the domain with a value of 10^{-4} s^{-1}. This corresponds to a beta-plane approximation at a latitude of $\theta = 45°$N. The experiments were performed with 30×30 grid points and a time step of 100s using a leap frog scheme.

The state vector $\mathbf{x}$, used in the filter algorithms, consists of the surface elevation $\mathbf{h}$ and the horizontal velocity components $\mathbf{u}$ and $\mathbf{v}$ at the grid points. The state dimension amounts to $n = 2700$. This number is sufficiently large to obtain meaningful filter results also for the low-rank algorithms, but it is still small enough to allow for a direct study of the filter-represented covariance matrices.

For the twin experiments the 'true' state trajectory of the system is generated by initializing with the state shown in the left panel of figure 4.1. It is in geostrophic balance and has a shape that ensures nonlinear evolution with the shallow water equations. Synthetic observations of the surface elevation at each grid point are generated by adding normally distributed random numbers of variance 10^{-4} m^2 to the true surface elevation. Using only the surface elevation as observations, the dimension of the observation vector is $m = 900$. The generated observations are quite accurate in comparison to the amplitude of the true surface elevation. This is useful, since the dependence of filtering performance on ensemble size can be better accessed for large ensembles with accurate observations. In the twin experiments it is assumed that the model is exact, thus no model error is simulated.

[1] We use the notation $\vec{u}$ for a spatially continuous vector field. The discretization of a field h, which is represented as a vector, is denoted by $\mathbf{h}$.

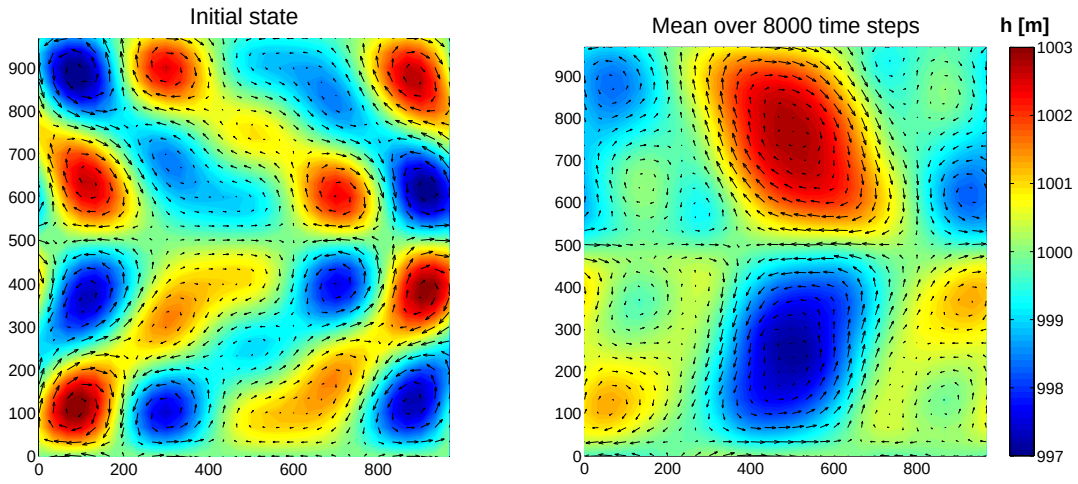


Figure 4.1: Surface elevation and velocity field of the true initial state (left) and mean state over 8000 time steps using each 10th step (right).

Two types of experiments are performed. For the first one, referred to as experiment 'A', the initialization of the model state estimate $\mathbf{x}_0^a$ and the corresponding covariance matrix $\mathbf{P}_0^a$ is performed for all three filter algorithms by applying the EOF procedure described by Pham et al. [68] which uses a sequence of model states. The initial state estimate $\mathbf{x}_0^a$ is chosen as the mean state of the true model simulation over 8000 time steps using each 10th time step. It is shown in the right panel of figure 4.1. The covariance matrix $\mathbf{P}_0^a$ is computed as the variation of the true model trajectory about this mean. This matrix does not reflect the estimated error of the initial state but the estimated mean temporal variability of the model state. The procedure, however, yields a consistent and simple way to obtain variance estimates together with estimates of the covariances.

This mean and covariance matrix serve as a baseline. However, it soon turned out that all algorithms can improve this "state of large ignorance". A much more enlightening setting would be to use a model state and covariance matrix that are already quite accurate and difficult to improve. To this end, the initialization of the second type of experiments, referred to as experiment 'B', is conducted with the estimated state and covariance matrix after the second analysis update from an assimilation experiment of type A with the EnKF using a very large ensemble of $N = 5000$ members. This is a very accurate state estimate whose rms deviation from the true state is two orders of magnitude smaller than the initial estimate of type A. The structure of this state is thus very similar that of the true initial state displayed in the left panel of figure 4.1. In addition, the covariance matrix of type B is an estimated error covariance matrix of the state estimate. It has a strongly different structure compared with the covariance matrix of type A. This is obvious from the eigenvalue spectrum, displayed in figure 4.2. For type A the covariance matrix is ill-conditioned and the ten largest eigenmodes already explain 99% of the variance. In contrast to this, 371 eigenmodes are required to explain 90% of the variance for type B.

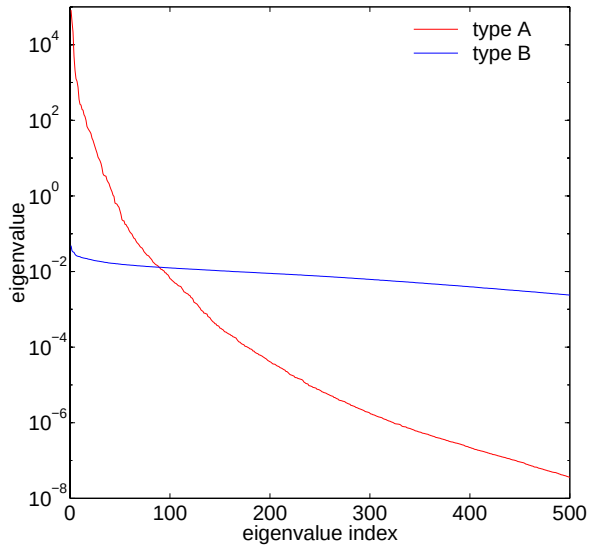


Figure 4.2: Eigenvalues for the covariance matrices for experiments of type A and B up to eigenvalue index 500.

Decomposed low-rank approximations $\hat{\mathbf{P}}_0^a = \mathbf{V}_0\mathbf{U}_0\mathbf{V}_0^T$ of the covariance matrix $\mathbf{P}_0^a$ are required to initialize the SEEK and SEIK filters. These are computed by incomplete eigenvalue decompositions of $\mathbf{P}_0^a$ retaining only the r largest eigenmodes. The N ensemble states required for the EnKF algorithm have been generated from the state estimate $\mathbf{x}_0^a$ and the covariance matrix $\mathbf{P}_0^a$ by a transformation of independent random numbers. For this, the eigenvalue decomposition of $\mathbf{P}_0^a$ is computed, yielding $\mathbf{P}_0^a = \mathbf{V}\mathbf{U}\mathbf{V}^T$. The eigenvectors are scaled by the square root of the corresponding eigenvalue as $\mathbf{L} = \mathbf{V}\mathbf{U}^{1/2}$. For each ensemble state $\{\mathbf{x}_0^{a(\alpha)},\ \alpha = 1, \ldots, N\}$ each scaled eigenvector $\mathbf{L}^{(i)}$ is multiplied by a random number $b_i^{(\alpha)}$ from a normal distribution of zero mean and unit variance and added to the state estimate $\mathbf{x}_0^a$:

$$\mathbf{x}_0^{a(\alpha)} = \mathbf{x}_0^a + \sum_{i=1}^{q} b_i^{(\alpha)}\ \mathbf{L}^{(i)}; \quad \alpha = 1, \ldots, N \tag{4.3}$$

Since the prescribed covariance matrix has a maximum rank of 799, we use only $q = 799$ eigenmodes in equation (4.3).

The assimilation experiments are performed over an interval of 8000 time steps for type A and 7600 time steps for type B with an analysis phase each 200 time steps. For a particular ensemble size N the rank in SEEK and SEIK is set to $r = N - 1$. In this case the number of model evaluations is equal for all three filter algorithms and the filtering performances can be directly related to computing time. Below the expression "ensemble size" is used to denote the number of different model states to be evolved. It will be equal to N for the EnKF and $r + 1$ for the SEEK and SEIK algorithms.

4.3 Comparison of Filtering Performances

To evaluate the filtering performance of the three algorithms the estimation error E_1, given by the rms deviation of the assimilated state from the true state, is considered separately for the three state fields **h**, **u**, and **v**. For the EnKF figure 4.3 shows estimation errors for experiments of type A with the three ensemble sizes r =30, 100, and 500. In addition, E_1 for an experiment conducting an evolution of the initial state estimate without assimilation is displayed. This free evolution shows only small variations in E_1 over assimilation time.

The temporal development of E_1 in the experiments with assimilation is characterized by a large reduction at the first analysis phase. This is due to an initially large error in the state estimate in connection with quite accurate observations. Subsequent analyses have significantly smaller influence. The EnKF algorithm performs better with increasing ensemble size where E_1 is strongly diminished. For small ensembles, like $N = 30$, E_1 increases with assimilation time, showing that the filter is unstable. As is visible in figure 4.3 the state estimate of the assimilation after 8000 time steps with 40 analysis cycles is even worse than without assimilation. For larger ensembles the assimilated state remains close to the true state.

Since only observations of the height field **h** are assimilated, the velocities are merely updated via cross covariances between the height field and the velocities. The representation of these covariances is generally worse than that of the height field variances and covariances as will be discussed in the following section. Due to this, the estimation errors E_1 normalized by the estimation errors of the free evolution are larger for the velocity components **u**, **v** than for the height field.

For the SEEK and SEIK filters, the general behavior of the estimation error in dependence on assimilation time and ensemble size is analogous to that of the EnKF. In order to compare the performance of all three filter algorithms in a compact way we define the normalized time integrated state estimation error by

$$E_2 := \frac{1}{3} \sum_{f=\mathbf{h},\mathbf{u},\mathbf{v}} \left(\sum_{k=k_{min}}^{40} \frac{E_1^{ass}(f,t_k)}{E_1^{free}(f,t_k)} \right) \tag{4.4}$$

where $E_1^{ass}(f,t_k)$ denotes the value of E_1 at time t_k for the state field $f \in \{\mathbf{h},\mathbf{u},\mathbf{v}\}$ from an assimilation experiment. $E_1^{free}(f,t_k)$ denotes the corresponding value for the free evolution. The summation over the analysis times excludes the initial state estimate since it would dominate the value of E_2 due to the large error decrease at the first analysis phase. Dependent on the type of experiment the summation starts at $k_{min} = 1$ for type A and $k_{min} = 3$ for type B. E_2 provides a rms measure of the decrease in estimation error due to data assimilation which respects a possible different scaling of the state fields.

Figure 4.4 shows E_2 for the three filter algorithms in dependence on ensemble size N for experiments of type A. For the EnKF and the SEIK algorithms mean results and standard deviations over 20 experiments with different random numbers used in the initialization phase are shown. There are significant variations of the filtering performance

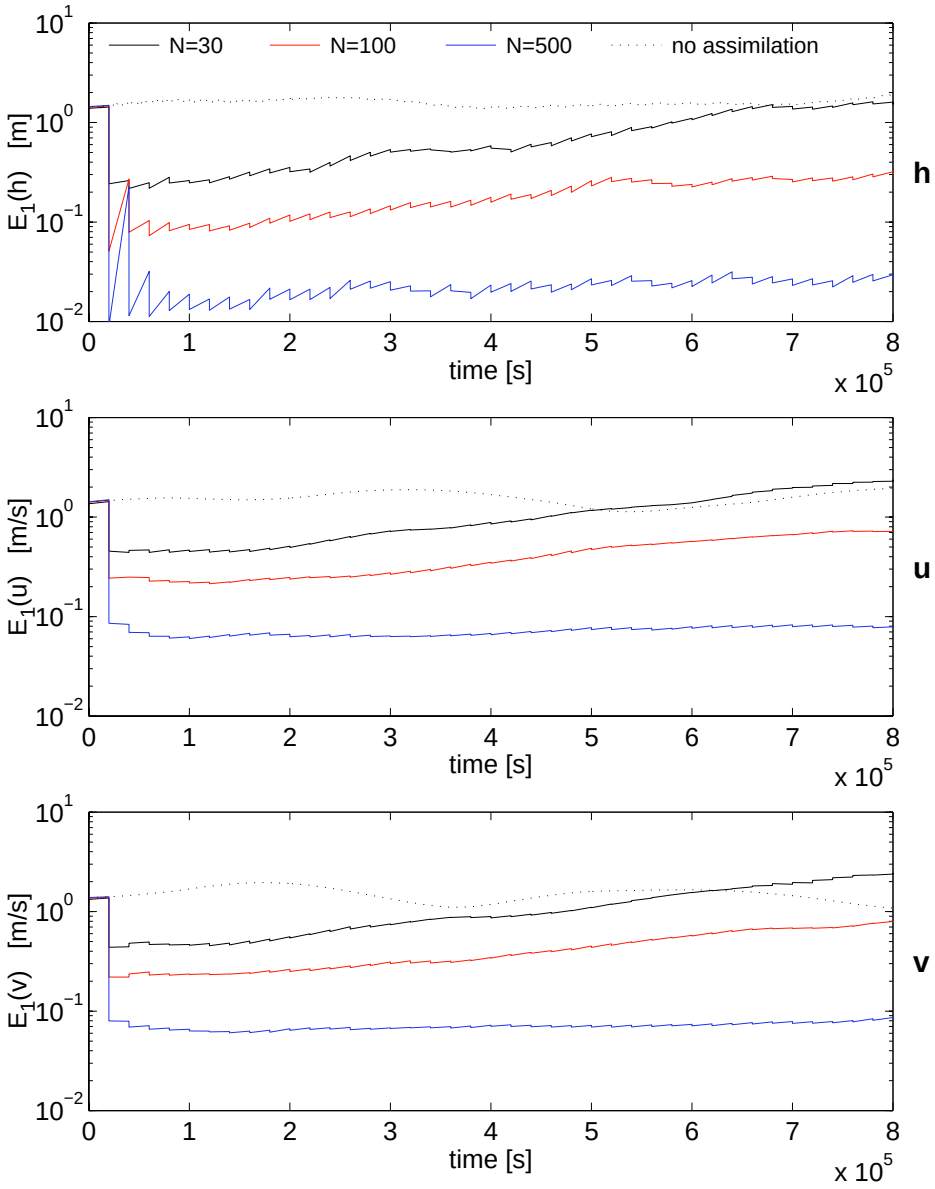


Figure 4.3: Estimation errors E_1 for experiments of type A. Shown is the time development of E_1 of the assimilated state for the EnKF for three ensemble sizes (N=30, 100, 500) and for a model simulation without assimilation.

depending on the used set of random numbers since the computer generated random numbers in fact do not represent the prescribed statistics exactly and do determine in which directions of the state space the ensemble vectors point. For small N the latter will likely lead to different qualities of the forecast ensemble. The SEEK algorithm is deterministic in its initialization, hence only the result of a single simulation per ensemble size is shown. As the observations are also generated using computer generated random numbers, they will also determine the filtering performance. This is of no concern here, since the observation error is quite small in the experiments and all three algorithms use the same observations.

Overall E_2 converges in the same manner for the EnKF and SEIK filters. A different convergence for SEIK which should be expected because of the second order exact sampling is not visible. This is caused by the eigenvalue spectrum of the covariance matrix $\mathbf{P}_0^a$ which shows that the number of significant eigenvalues is extremely small. For EnKF and SEIK, the convergence in the interval $100 < N < 500$ can be approximated by $E_2 \propto N^{-x}$ with $x \approx 1.2$ for the EnKF and $x \approx 1.0$ for the SEIK algorithm. Depending on the ensemble size, the mean values of E_2 for the EnKF are between 1.5 and 1.85 times larger than those for the SEIK filter. This also shows that, to achieve the same filtering performance, the ensemble for the EnKF needs to be between about 1.5 and 1.8 times larger than for the SEIK. These numbers are of course specific for the configuration of these experiments. However, variations of the assimilation interval and strong increase of the rms errors in the observations by a factor of 100 preserved the relative performances of the three algorithms. The behavior for the SEEK deviates significantly from that of the EnKF and SEIK. For $N < 70$ the SEEK filter shows the best filtering performance of the three algorithms. But, with further increasing ensemble size, E_2 stagnates at a rather large value. The reason for this behavior is further examined in section 4.4.

For experiments of type B with the EnKF, the estimation error E_1 over time is displayed in figure 4.6. Here the initial state approximates the true state quite well but without assimilation the rms deviation increases by about two orders of magnitude until the final time step. Thus, the conditions for this experiment are quite different from those of type A in which the initial state estimate was strongly deviating from the true state and the free evolution remained over simulation time at an almost constant rms deviation from the true state. In the experiments of type B the assimilation of height field observations keeps the estimates of all state fields much closer to the true state compared with the simulation without assimilation. As for type A, the estimation error of the velocity components is higher that for the sea level.

The error measure E_2 is displayed in figure 4.5 in dependence on ensemble size for the experiments of type B. Here mean results and standard deviations over 20 experiments with different random numbers in the initialization are only shown for the EnKF. The dependence of the SEIK filter on the random numbers used in the initialization is negligible for this type of experiment (data not shown). The performance of SEEK and SEIK is almost indistinguishable, with a relative difference of the values of E_2 below $6 \cdot 10^{-3}$. The values of E_2 are smaller for type B than for type A which is due to the normalization by E_1^{free} when computing E_2. Since E_1^{free} increases strongly over

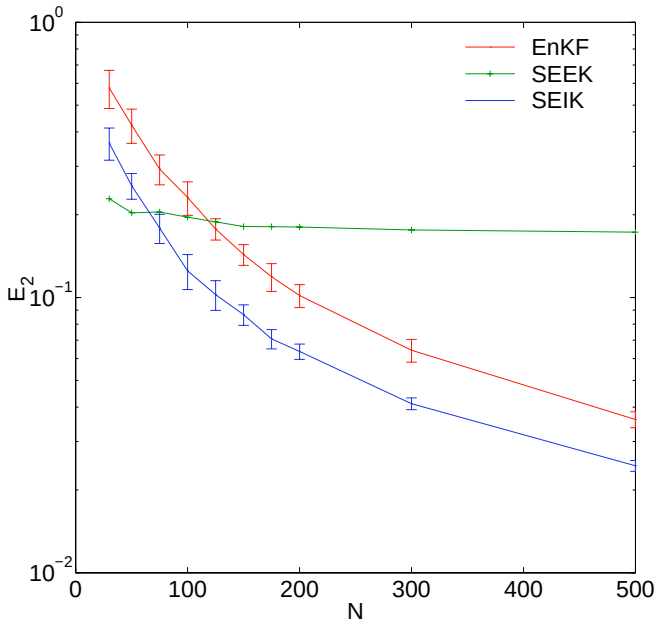


Figure 4.4: Normalized time integrated estimation errors E_2 for the three filter algorithms in dependence on the ensemble size N ($N = r + 1$ for SEEK and SEIK) for experiments of type A. For EnKF and SEIK mean values and standard deviations over 20 experiments for each ensemble size are show. Each experiment used different random numbers for the ensemble initialization.

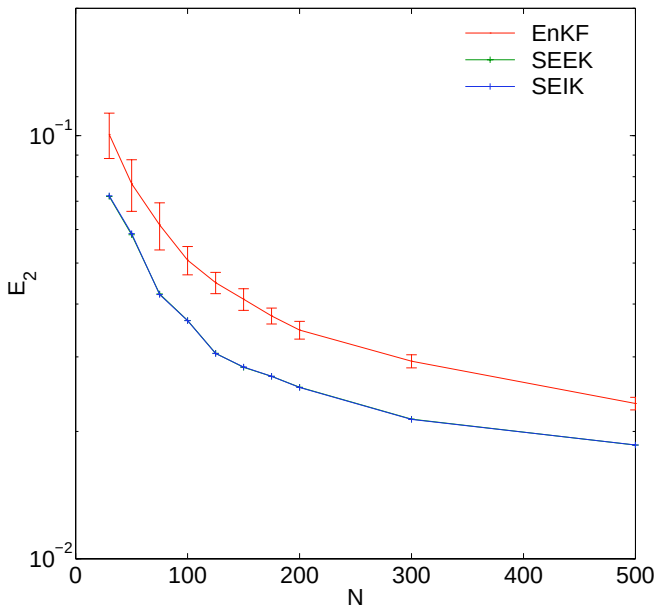


Figure 4.5: Normalized time integrated estimation errors E_2 analogous to figure 4.4 for experiments of type B. For EnKF mean values and standard deviations over 20 experiments are shown analogous to figure 4.4. The lines of SEEK and SEIK lie on top of each other.

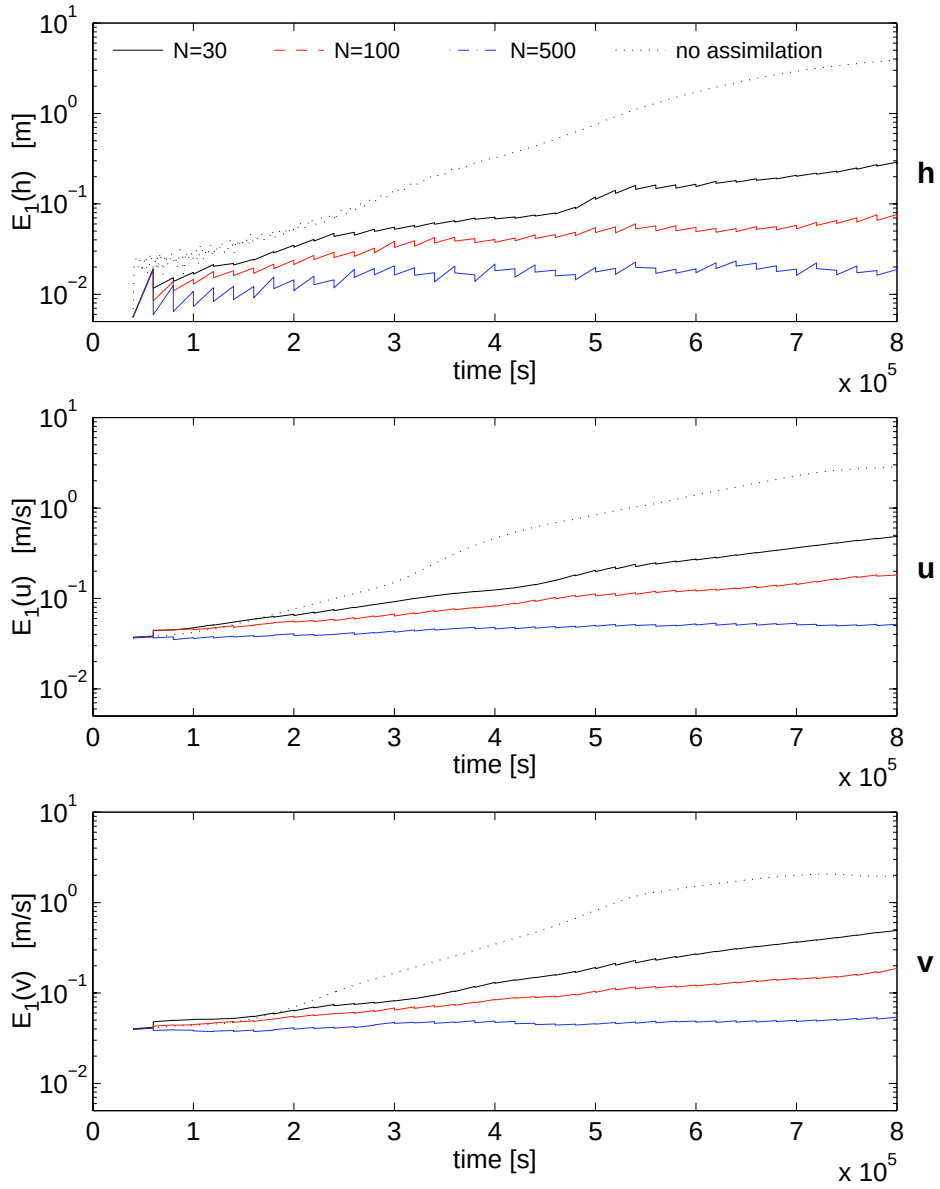


Figure 4.6: Estimation errors E_1 for experiments of type B. Shown is the time development of E_1 of the assimilated state for the EnKF for three ensemble sizes (N=30, 100, 500) and for a model simulation without assimilation.

time the normalization returns smaller values than in experiments of type A in which E_1^{free} remained almost constant. As for type A the value of E_2 converges similarly for the EnKF and SEIK filters. But for small ensembles ($N \leq 75$) SEIK converges faster than EnKF. Again the dependence of E_2 on N can be approximated in the interval $100 < N < 500$ to be $E_2 \propto N^{-x}$ with $x \approx 0.42$ for the EnKF and $x \approx 0.44$ for the SEIK algorithm. Thus, the convergence with ensemble size is much smaller for type B than for type A. To obtain the same filter performance, the ensemble in the EnKF would need to be between about 1.6 and 2.2 times larger than for SEIK. This result corresponds to that reported by Heemink et al. [31]. There the RRSQRT filter, which is similar to the SEEK algorithm as was discussed in section 2.4.1, yielded comparable estimation errors to the EnKF for about half the number of model evaluations.

According to the discussion on the initialization of EnKF and SEIK in section 3.2, it is possible to interchange the methods of Monte Carlo sampling and second order exact sampling between these two filters. Figure 4.7 shows a comparison of SEIK

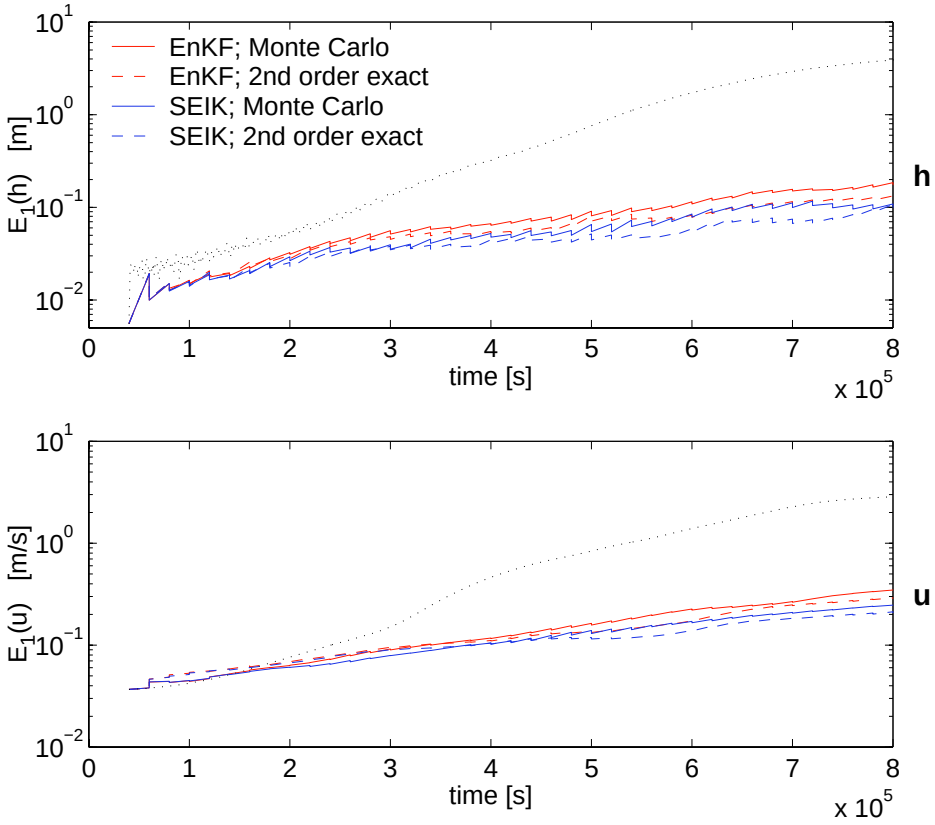


Figure 4.7: Comparison of the estimation errors E_1 for SEIK and EnKF for experiments of type B with their typical initialization and with interchanged initializations for an ensemble size of $N = 50$. The dotted line shows E_1 for a model evolution without assimilation. The behavior of E_1 for the zonal velocity component **v** is similar to that of **u** and hence not shown.

and EnKF with interchanged initializations for experiments of type B with $N = 50$. The experiments of both types yield a 5 to 10% better filtering performance for the EnKF algorithm when the filter is initialized by second order exact sampling instead of pure Monte Carlo sampling. The performance of the SEIK filter degrades by about the same amount if the Monte Carlo initialization is applied. After interchanging the initialization the SEIK filter still performs better than EnKF. This is caused by the introduction of noise into the ensemble by the observation ensemble required in the analysis scheme of the EnKF algorithm as will be discussed below.

4.4 Statistical Examination of Filtering Performance

To gain insight into the reasons for the different filtering performances of the three algorithms, an examination of the sampling quality of the represented state covariance matrices is performed in the sequel. At first, some additional analysis quantities are defined. Based on these quantities it is then discussed how the different variants of forecasting and different choices of ensembles can lead to estimates of the covariance matrix, and hence the error subspace, of strongly different quality.

4.4.1 Definition of Analysis Quantities

To define analysis quantities measuring the sampling quality, let us reconsider the filter algorithms. The SEEK filter evolves the state estimate with the nonlinear dynamic model and the eigenmodes of the low-rank approximated state covariance matrix with the linearized dynamic model or a gradient approximation of it. The EnKF and SEIK filters both evolve an ensemble of model states with the nonlinear dynamic model. The capability of the forecast phase to provide a realistic representation of the error subspace is reflected by the sampling quality of the state covariance matrix $\mathbf{P}$.

To discuss the analysis phase we consider the covariance matrix to consist of sub-matrices as:

$$\mathbf{P} = \begin{pmatrix} \mathbf{P}_{hh} & \mathbf{P}_{hu} & \mathbf{P}_{hv} \\ \mathbf{P}_{uh} & \mathbf{P}_{uu} & \mathbf{P}_{uv} \\ \mathbf{P}_{vh} & \mathbf{P}_{vu} & \mathbf{P}_{vv} \end{pmatrix} \tag{4.5}$$

Here the sub-matrices $\{\mathbf{P}_{ij} = \mathbf{P}_{ji}^T\}$ are $n/3 \times n/3$ matrices with $\mathbf{P}_{hh}$, $\mathbf{P}_{uu}$, and $\mathbf{P}_{vv}$ containing respectively the covariances of the height field and the two velocity components. The off-diagonal sub-matrices $\{\mathbf{P}_{ij},\ i \neq j\}$ contain the cross covariances between different state fields. The measurement operator projects a state vector onto its height field part, thus

$$\mathbf{H} = (\mathbf{I}_{m\times m} \quad \mathbf{0}_{m\times 2m}) \tag{4.6}$$

where $\mathbf{I}$ is the identity matrix and $\mathbf{0}$ the matrix containing only zeros. In the experiments, all observations were assumed to be uncorrelated with variances of constant value var_{hh}. Thus the observation error covariance matrix is:

$$\mathbf{R} = \mathrm{var}_{hh}\mathbf{I}_{m\times m}\ . \tag{4.7}$$

With this specifications, the analysis equation for the state in SEEK and SEIK (respectively equation (2.29) or (2.68)) simplifies to

$$\mathbf{x}^a = \mathbf{x}^f + \mathrm{var}_{hh}^{-1} \begin{pmatrix} \mathbf{P}_{hh}^a \\ \mathbf{P}_{uh}^a \\ \mathbf{P}_{vh}^a \end{pmatrix} \mathbf{d} \tag{4.8}$$

with observation-state residual, sometimes also called innovation, $\mathbf{d} = \mathbf{y}^o - \mathbf{h}^f$ where $\mathbf{h}^f$ is the estimated forecast height field. For the EnKF the analysis equation (2.41) for the ensemble states is also valid for the ensemble mean, see [17]. In the case considered here it simplifies to

$$\mathbf{x}^a = \overline{\mathbf{x}^f} + \begin{pmatrix} \mathbf{P}_{hh}^f \\ \mathbf{P}_{uh}^f \\ \mathbf{P}_{vh}^f \end{pmatrix} \left[\mathbf{P}_{hh}^f + \mathrm{var}_{hh} \mathbf{I}_{m \times m} \right]^{-1} \mathbf{d} =: \overline{\mathbf{x}^f} + \mathbf{A}\mathbf{d} \ . \tag{4.9}$$

According to equations (4.8) and (4.9) only the covariances $\mathbf{P}_{hh}$ in the height field and the cross covariances $\mathbf{P}_{uh}$ and $\mathbf{P}_{vh}$ between height field and the velocity components are considered in the analysis update of the state estimate. The other sub-matrices are as well updated during the analysis update of the covariance matrix and all parts of $\mathbf{P}$ determine the quality of the forecast.

To compare the three filter algorithms despite their different analysis equations we define update matrices $\mathbf{B}$. For the SEEK and SEIK filters we define the elements $\{\mathbf{B}_{(\alpha,\beta)}, 1 \le \alpha \le n, 1 \le \beta \le m\}$ by

$$\mathbf{B}_{(\alpha,\beta)}^a := \mathrm{var}_{hh}^{-1} \mathbf{P}_{(\alpha,\beta)}^a \mathbf{d}_{(\beta)} \ . \tag{4.10}$$

For the EnKF the definition is analogously

$$\mathbf{B}_{(\alpha,\beta)}^f := \mathbf{A}_{(\alpha,\beta)} \mathbf{d}_{(\beta)} \ . \tag{4.11}$$

The update matrices $\mathbf{B}$ correspond to the matrix-vector products in equations (4.8) and (4.9) without performing the summation. For the SEEK and SEIK filters this amounts to a scaling of the covariances by the elements of the residual vector. Thus, the update matrices take into account not only the different sampling qualities of the state covariance matrix but also different residuals $\mathbf{d}$. Accordingly, an estimate of the analysis quality for the single state fields will be provided by the sampling quality of the sub-matrices $\mathbf{B}_{hh}$, $\mathbf{B}_{uh}$, and $\mathbf{B}_{vh}$.

To quantify the sampling quality we compare the computed update matrices with an update matrix obtained from an EnKF assimilation experiment with ensemble size $N = 5000$, referred to as the "ideal" update matrix $\mathbf{B}^{ideal}$. For the comparison we compute correlation coefficients $\rho_{\mathbf{B}}$ between the sampled and ideal update sub-matrices and regression coefficients $\beta_{\mathbf{B}}$ from the ideal to the sampled update sub-matrices. We focus on the very first analysis phase in which for experiments of type A the largest reduction of the estimation errors occurs.

4.4.2 The Influence of Ensemble Size in Type A

In table 4.1 experiments of type A are examined for assimilation with an ensemble size $N = 30$. Displayed are the correlation and regression coefficients $\rho_{\mathbf{B}}$, $\beta_{\mathbf{B}}$ for the height field $\mathbf{h}$ and the zonal velocity component $\mathbf{u}$. The coefficients for the meridional velocity component $\mathbf{v}$ are similar to those for $\mathbf{u}$ and thus not shown. In addition the relative estimation error

$$E_3(f) = \frac{E_1^{ass}(f, t_1)}{E_1^{free}(f, t_1)} \tag{4.12}$$

after the first analysis is shown for the fields $f \in (\mathbf{h}, \mathbf{u})$. For comparison, the values of E_3 for the ideal experiment are much smaller with $E_3(\mathbf{h}) = 0.005$ and $E_3(\mathbf{u}) = 0.04$. Thus, the filtering performance will increase strongly with growing ensemble size and the improvement will be larger for the height field than for the velocity components.

The order of the values of E_3 for the three filters is the same as that of the time integrated E_2 values for $N = 30$ displayed in figure 4.4. The SEEK has the smallest value of E_3, followed by SEIK and then EnKF. The ratio of the time integrated E_2 for the EnKF to that of the SEIK is 1.59. It is larger than the corresponding ratio of E_3 values after the first analysis update which is 1.24. This is caused by the use of an observation ensemble in the analysis of the EnKF which destabilizes the assimilation process. This will be examined in more detail below.

The correlation and regression coefficients $\rho_{\mathbf{B}}$, $\beta_{\mathbf{B}}$ reflect the different filtering performances of the first analysis update. Overall it is visible that there is a significant correlation between the sampled and the ideal sub-matrices. The small regression coefficients show in addition that the amplitudes are strongly underestimated. Using in the experiments observations with larger errors decreases the amount of underestimation (data not shown). The underestimation is even more pronounced when one considers only the correlation and regression coefficients for the variance part, i.e. the diagonal, of the height field update sub-matrix. These coefficients are also shown in table 4.1, denoted as ρ_{var} and β_{var}. For $N = 30$ the correlation coefficients ρ_{var} are already very near to unity. The regression coefficients β_{var} show, however, a very strong underesti-

Table 4.1: Examination of the sampling quality at first analysis phase for experiments of type A with $N = 30$. Shown are relative estimation errors E_3 and the correlation $\rho_{\mathbf{B}}$ and regression $\beta_{\mathbf{B}}$ coefficients between the ideal and sampled update sub-matrices for the height field $\mathbf{h}$ and the zonal velocity $\mathbf{u}$. In addition, the correlation ρ_{var} and regression β_{var} coefficients of the variance part for the height field are shown.

	field	E_3	$\rho_{\mathbf{B}}$	$\beta_{\mathbf{B}}$	ρ_{var}	β_{var}
EnKF		0.168	0.305	0.091	0.961	0.071
SEEK	$\mathbf{h}$	0.089	0.325	0.107	0.959	0.086
SEIK		0.135	0.320	0.107	0.959	0.084
EnKF		0.309	0.126	0.015		
SEEK	$\mathbf{u}$	0.179	0.188	0.035		
SEIK		0.273	0.130	0.017		

Table 4.2: Examination of the sampling quality at the first analysis for experiments of type A with $N = 200$. Shown are the same quantities as in table 4.1.

	field	E_3	$\rho_{\mathbf{B}}$	$\beta_{\mathbf{B}}$	ρ_{var}	β_{var}
EnKF		0.015	0.756	0.570	0.996	0.477
SEEK	**h**	0.035	0.554	0.277	0.988	0.227
SEIK		0.012	0.756	0.598	0.995	0.503
EnKF		0.103	0.502	0.315		
SEEK	**u**	0.191	0.324	0.121		
SEIK		0.081	0.496	0.332		

mation of the variance. In the experiments, the structure of the update sub-matrix $\mathbf{B}_{hh}$ corresponding to a single grid point, as well as the covariance sub-matrix $\mathbf{P}_{hh}$, consists of noise of rather low amplitude and a significantly larger peak with a radius of about two grid points around the location of the specified grid point. Thus the variance will dominate the analysis while most of the noise will average out when computing the product $\mathbf{P}_{hh}\mathbf{d}$. For the EnKF the smaller values of $\rho_{\mathbf{B}}$ and $\beta_{\mathbf{B}}$ for $\mathbf{h}$ point to the fact that here the analysis is less accurate than for SEEK and SEIK. This is confirmed by the value of E_3 which is larger for the EnKF than for the two other filters. For the difference between SEEK and SEIK this is less obvious.

For the velocity components the sampling quality of $\mathbf{B}$ is generally worse than for the height field. This is due to the fact that only $\mathbf{h}$ is observed and $\mathbf{u}$, $\mathbf{v}$ are updated via the covariance sub-matrices $\mathbf{P}_{uh}$ and $\mathbf{P}_{vh}$. These have a structure with multiple extrema and are more difficult to sample than the variance-dominated $\mathbf{P}_{hh}$ (data not shown). For all three filters the values of $\rho_{\mathbf{B}}$ and $\beta_{\mathbf{B}}$ are nearest to unity in the case of the SEEK algorithm. This is consistent with the filter's small value of E_3. In experiments of type A the SEEK filter is able to sample the sub-matrices $\mathbf{P}_{uh}$ and $\mathbf{P}_{vh}$ for small ensembles significantly better than the SEIK and EnKF filters.

For $N = 200$ the sampling quality of the update matrices is examined in table 4.2. Compared with $N = 30$ the estimation errors E_3 after the first analysis are much smaller. This decrease is minor for the velocity components than for the height field due to the worse sampling of cross correlations between $\mathbf{h}$ and the velocity components $\mathbf{u}$, $\mathbf{v}$. The increased regression coefficients $\beta_{\mathbf{B}}$ show that the underestimation of the correlations has diminished. In addition, according to the increased correlation coefficients $\rho_{\mathbf{B}}$ and ρ_{var}, covariances as well as variances are sampled much more realistic. The similarity of the coefficients for SEIK and EnKF has increased compared with $N = 30$, but the SEIK still shows the better sampling quality.

The estimation error measures E_2 and E_3 for $N = 200$ are larger for the SEEK filter than for the SEIK and EnKF filters. This is consistent with the values of $\rho_{\mathbf{B}}$ and $\beta_{\mathbf{B}}$ which are smaller for the SEEK than for the two other filters. This inferior sampling quality of SEEK is caused by the direct forecast of the eigenmodes of the state covariance matrix $\mathbf{P}$. The modes with larger index represent gravity waves. These are impossible to control by the data assimilation in our experimental setup. Hence, these

modes do not provide any useful information to the error subspace and the filtering performance stagnates. For the estimated velocity components the experiments show that this can even lead to a small decrease in the filtering performance for increasing N.

4.4.3 Sampling Differences between EnKF and SEIK

The different sampling quality of the EnKF and SEIK filters is due to the distinct variants to generate the ensembles in both algorithms. Interchanging the initialization methods between the algorithms results, at the first analysis phase, in an exchange of the values of E_3, $\rho_{\mathbf{B}}$, and $\beta_{\mathbf{B}}$. Using the same ensemble and neglecting model errors, both filters are equivalent during the first analysis phase with respect to the update of the state estimate since the predicted error subspaces are identical. Such an equivalence does not exist for the update of $\mathbf{P}$ due to the implicit update of this matrix in the EnKF algorithm. While the update of $\mathbf{P}$ for the Extended Kalman filter is described by equation (2.16) the update of $\mathbf{P}$ for the EnKF algorithm is given implicitly by

$$\tilde{\mathbf{P}}^a = (\mathbf{I} - \mathbf{KH})\tilde{\mathbf{P}}^f(\mathbf{I} - \mathbf{K}^T\mathbf{H}^T) + \mathbf{K}\tilde{\mathbf{R}}\mathbf{K}^T + \mathcal{O}(< \delta\mathbf{x}^f(\delta\mathbf{y}^o)^T >) \; . \tag{4.13}$$

Here $\tilde{\mathbf{R}}$ is the observation error covariance matrix as sampled by the ensemble of observation vectors. $\tilde{\mathbf{P}}^f$, $\tilde{\mathbf{P}}^a$ are the covariance matrices of the forecast and analysis state ensembles. The last term $\mathcal{O}(< \delta\mathbf{x}^f(\delta\mathbf{y}^o)^T >)$ denotes the spurious covariances between the state and observation ensembles. In SEEK and SEIK this last term is zero and $\tilde{\mathbf{R}}$ is replaced by the prescribed matrix $\mathbf{R}$ and $\tilde{\mathbf{P}}$ denotes the rank-r approximated state covariance matrix. For SEEK and SEIK equation (4.13) reduces to the correct KF update equation for a covariance matrix $\tilde{\mathbf{P}}$. For the EnKF the sampled matrix $\tilde{\mathbf{R}}$ and the correlations between the state and observation ensembles insert noise into the analysis ensemble which represents the state covariance matrix. Whitaker and Hamill [94] discussed this effect in a simple one-dimensional system. In order to quantify the introduction of noise the two definitions (4.10) and (4.11) of $\mathbf{B}$ can be examined. Without sampling errors, both definitions are equally valid. Thus for the SEEK and SEIK filters the update matrices computed from either equation are identical. For the EnKF the resulting update matrices are different.

In table 4.3 the coefficients $\rho_{\mathbf{B}}$ and $\beta_{\mathbf{B}}$ for update matrices computed with equations (4.10) or (4.11) are compared for the EnKF algorithm with $N = 30$ for experiments of type A. The values of $\rho_{\mathbf{B}}$ computed from the forecast covariances according to equation (4.11) are about 1.5 times larger compared with those computed with equation (4.10) from the analysis covariances. Despite this, the regression coefficients $\beta_{\mathbf{B}}$ remain almost unchanged. Also the coefficients ρ_{var} and β_{var} show an analogous but much smaller ratio. The introduction of noise to the ensemble states at each analysis phase leads to more unstable forecasts in the EnKF in comparison to the SEIK. Over the course of the assimilation process the estimation error E_1 deviates increasingly for the two filters. This leads to the values of E_2 shown in figure 4.4 in which the difference in filtering performance between EnKF and SEIK is larger than just for the first analysis.

Table 4.3: Comparison of the sampling quality of the update sub-matrices for the EnKF with $N = 30$ for experiments of type A. Shown are correlation $\rho_{\mathbf{B}}$ and regression $\beta_{\mathbf{B}}$ coefficients for sampled update sub-matrices computed from the forecast covariance matrix ($\mathbf{B}^f$, equation (4.11)) and from the analysis covariance matrix ($\mathbf{B}^a$, equation (4.10)). In addition, the correlation and regression coefficients (ρ_{var}, β_{var}) for the variance part of the height field update sub-matrix are shown.

B computed by	field	$\rho_{\mathbf{B}}$	$\beta_{\mathbf{B}}$	ρ_{var}	β_{var}
$\mathbf{B}^f_{(\alpha,\beta)} = \mathbf{A}_{(\alpha,\beta)}\mathbf{d}_{(\beta)}$	**h**	0.305	0.091	0.961	0.071
$\mathbf{B}^a_{(\alpha,\beta)} = \mathrm{var}^{-1}_{hh}\mathbf{P}^a_{(\alpha,\beta)}\mathbf{d}_{(\beta)}$	**h**	0.207	0.093	0.937	0.072
$\mathbf{B}^f_{(\alpha,\beta)} = \mathbf{A}_{(\alpha,\beta)}\mathbf{d}_{(\beta)}$	**u**	0.126	0.015		
$\mathbf{B}^a_{(\alpha,\beta)} = \mathrm{var}^{-1}_{hh}\mathbf{P}^a_{(\alpha,\beta)}\mathbf{d}_{(\beta)}$	**u**	0.082	0.014		

4.4.4 Experiments with the Idealized Setup (Type B)

The sampling quality of the update matrices for experiments of type B for ensembles of size $N = 30$ and $N = 200$ are respectively shown in tables 4.4 and 4.5. For the SEEK and SEIK filters the values of E_3, $\rho_{\mathbf{B}}$, and $\beta_{\mathbf{B}}$ for are identical for **h** and almost identical for **u** and **v** for both ensemble sizes. Thus, the SEEK filter shows no problem caused by the mode forecasts in this type of experiment. This can be related to the different structure of the covariance matrix which leads to mode forecasts which provide realistic directions of the error subspace even for high eigenvalue indices. For **h** the EnKF shows a slightly larger estimation error E_3 than SEIK. This corresponds to the smaller values of $\rho_{\mathbf{B}}$ which show that the update matrices are less realistic sampled for the EnKF compared with the SEIK. The EnKF, however, underestimates the amplitude of the covariances to a lesser degree than SEIK does. The variance part of the update matrices is represented better by the EnKF than by SEIK as is visible from both the values of ρ_{var} and β_{var}. The smaller regression coefficients in the case of the SEIK filter result from the low-rank approximation of the matrix **P** which systematically underestimates the overall variance. Due to the structure of **P** in experiments of type B, as discussed in section 4.2, the disregarded variance is non-negligible here even for $N = 200$.

The velocity components are much worse filtered here than in the experiments of type A. For $N = 30$ the values of E_3 even increase showing that the sampled covariances are not realistic. For $N = 200$ a small decrease of the estimation error is visible which is stronger for the SEIK compared with the EnKF. Since the ideal values of E_3 are 0.2 for **h** and 0.75 for **u** there will be no strong decrease in E_3 any more for larger ensembles. Over the whole assimilation period the performance of all three filters is however better than at the first analysis phase. While the non-assimilated state diverges from the true state, the data assimilation keeps the estimation error almost constant. This leads to the small values of the time integrated estimation error E_2 displayed in figure 4.5.

Table 4.4: Examination of the first analysis for experiments of type B with $N = 30$. Shown are the same quantities as in table 4.1.

	field	E_3	$\rho_{\mathbf{B}}$	$\beta_{\mathbf{B}}$	ρ_{var}	β_{var}
EnKF		0.446	0.408	0.206	0.973	0.150
SEEK	**h**	0.431	0.425	0.171	0.944	0.119
SEIK		0.431	0.425	0.171	0.944	0.119
EnKF		1.045	0.175	0.090		
SEEK	**u**	1.135	0.366	0.213		
SEIK		1.137	0.367	0.213		

Table 4.5: Examination of the first analysis for experiments of type B with $N = 200$. Shown are the same quantities as in table 4.1.

	field	E_3	$\rho_{\mathbf{B}}$	$\beta_{\mathbf{B}}$	ρ_{var}	β_{var}
EnKF		0.273	0.802	0.703	0.996	0.630
SEEK	**h**	0.269	0.847	0.651	0.991	0.533
SEIK		0.269	0.847	0.650	0.991	0.532
EnKF		0.981	0.519	0.559		
SEEK	**u**	0.872	0.766	0.729		
SEIK		0.875	0.766	0.728		

4.5 Summary

The behavior of the SEEK, EnKF, and SEIK filters has been assessed utilizing identical twin experiments. The experiments applied a shallow water equation model with nonlinear evolution and assimilated synthetic observations of the sea surface elevation. Two types of experiments have been performed with distinct initializations of the state estimate and state covariance matrix. For identical initial conditions, the filter algorithms showed quite different abilities to reduce the estimation error. In addition, the filtering performances depended differently on the ensemble size.

Under some circumstances, the SEEK filter shows a distinct behavior from the two other algorithms caused by the direct evolution of modes of the state covariance matrix. This depends on the structure of this matrix. For the experiments of type A, in which the covariance matrix is dominated by a small number of large-scale modes, the performance of SEEK is different from that of EnKF and SEIK. For experiments of type B, in which the covariance matrix is variance dominated, SEEK and SEIK perform almost identical. The superior performance of SEEK for smallest ensemble sizes in experiments of type A appears to be by chance but shows that a mode-oriented filter algorithm can under some circumstances yield a superior filter performance than the ensemble based filters. SEEK is well suited to filter rather coarse structures in which nonlinearity is not pronounced.

The EnKF and SEIK algorithms show similar convergence with increasing ensemble size. The SEIK filter exhibits superior performance compared with the EnKF algorithm due to the initialization by minimum second order exact sampling of the low-rank approximated state covariance matrix. This sampling leads to a superior ensemble representation of this matrix, in particular, for small ensembles. In addition, the SEIK filter does not suffer from noise introduced into the state ensemble by an observation ensemble as required by the EnKF.

Statistical analyses of the quality of the sampled state covariance matrices showed how these matrices differ for the examined algorithms. The structure of the variances is in all filters quite well represented, but their amplitudes are underestimated. Dependent on the structure of the covariance matrix, the low-rank initialization used in SEEK and SEIK tends to underestimate the variances even more than the Monte-Carlo initialization used in EnKF. The sampling of the full covariance sub-matrices for the single state fields is inferior for all three filters in comparison to the variances. The representation of the covariances for the height field is significantly better than that of the cross correlations between the height field and the velocity components. This is due to the variance dominated structure of the height field covariances. The sampling quality of the covariances and cross correlations can be improved, at least for the SEIK and EnKF, by increasing the ensemble size.

Chapter 5

Summary

This part of this two-part work compared three filter algorithms based on the Kalman filter, namely the Ensemble Kalman Filter (EnKF), the Singular Evolutive Extended Kalman (SEEK) filter and the Singular Evolutive Interpolated Kalman (SEIK) filter. In the mathematical comparison, the unified interpretation of the filter algorithms as Error Subspace Kalman Filters (ESKF) was introduced. This interpretation is motivated by the fact that the three algorithms apply a low-rank approximation of the state covariance matrix used in the Extended Kalman filter (EKF). Hence, they approximate the error space of the EKF by a low-dimensional error subspace. In addition, the three filter algorithms apply the analysis equations of the EKF adapted to the respective algorithm. Thus, the analysis assumes Gaussian statistics of both the state estimate and the observations.

The SEEK and SEIK filters are typically initialized from a state estimate and a state covariance matrix which can be provided in some decomposed form, e.g. as a sequence of model states. The state covariance matrix is approximated by a matrix of low rank. This low-rank matrix is then exactly represented either by the eigenmodes of the matrix in the case of SEEK or by a random ensemble of minimal size in SEIK. The EnKF algorithm can also be initialized from a state estimate and a corresponding covariance matrix. This information is typically used to generate a random ensemble by Monte Carlo sampling. The statistics of the generated ensemble approximate the state estimate and the state covariance matrix.

In the forecast phase, the EnKF and SEIK filters are equivalent. Both perform a nonlinear ensemble forecast. In contrast to this, the SEEK filter forecasts explicitly the modes of the covariance matrix by the linearized model or a gradient approximation of it. The state estimate is explicitly evolved using the nonlinear model. It has been shown that the ensemble forecast performed in the EnKF and SEIK algorithms is better suited for nonlinear models than the forecast scheme used in the SEEK filter.

It has been shown that the analysis increment of all three filter algorithms is given by a weighted average of vectors which belong to the error subspace. The analysis phase of the EnKF algorithm is less efficient than that of the SEEK and SEIK filters if the amount of observations is larger than the ensemble size. This is due to the fact, that the EnKF algorithm uses the representer analysis variant which operates on the observation space. In contrast to the EnKF algorithm, the SEEK and SEIK filters

operate on the error subspace. Another apparent problem of the EnKF algorithm is that the analysis phase introduces noise to the state ensemble caused by a numerically generated ensemble of observation vectors which is required by the analysis scheme.

While the EnKF algorithm computes its new ensemble during the analysis phase, the SEEK and SEIK filters contain a resampling phase. Its has been shown that this will not render the latter two algorithms to be less efficient with respect to the required computation time than the EnKF.

Overall, the mathematical comparison showed that the SEEK filter is a re-formulation of the EKF for a low-rank state covariance matrix stored in decomposed form. It has the numerically most efficient analysis scheme of the three filter algorithms but shows only limited abilities to handle nonlinearity. The EnKF algorithm is a Monte Carlo method which is not designed to profit from the fact that the probability density of the model state will be at least approximately Gaussian. Thus, it is not explicitly considered that the density can be represented by a linear error space which can be approximated by its major directions. SEIK filter takes this into account and approximates the covariance matrix, which characterizes the error space, by a low-rank matrix. Hence, the SEIK filter has the same ability to treat nonlinearity as the EnKF algorithm but a more efficient analysis scheme. The EnKF algorithm can be expected to exhibit an enhanced filtering performance when it is initialized from a low-rank covariance matrix analogous to the SEIK filter. The problem of noise introduction by the observation ensemble will, however, remain.

The theoretical findings have been confirmed by numerical experiments using a shallow water equation model with nonlinear evolution. In identical twin experiments, synthetic observations of the sea surface elevation have been assimilated. The experiments have been interpreted in terms of the estimation errors and by a statistical analysis of the sampling quality of the state covariance matrices. The experiments showed that the SEIK algorithm is an ensemble algorithm comparable to the EnKF with the benefit of a very efficient scheme for analysis and resampling. In addition, the SEIK filter does not suffer from noise introduced into the state ensemble by an observation ensemble as required by the EnKF. As the EnKF and SEIK filters, the SEEK algorithm is able to provide good state estimates. The SEEK filter is, however, sensitive to the mode vectors it needs to evolve. Due to this, the SEEK filter can exhibit a distinct filtering behavior from the EnKF and SEIK filters. In the experiments this depended on the structure of the state covariance matrix. In general, it will also depend on the physical system which is simulated. The SEEK filter will be, however, well suited to filter rather coarse structures in which nonlinearity is not pronounced. The experiments also showed that initialization methods using higher order sampling schemes like the second order exact sampling are appealing due to the better representation of the state covariance matrix, in particular for small ensembles.

The experiments performed here are of course highly idealized. For example, an inclusion of model error would be desirable. But, for the EnKF and SEIK filters, it can be expected that this will not lead to significant changes in the relative filter performance, since both algorithms can treat the model error in the same way. Results obtained with more realistic experiments will be discussed in chapter 9 where the filter algorithms are applied to the three-dimensional finite element ocean model FEOM.

Part II

Parallel Filter Algorithms

Chapter 6

Overview and Motivation

The development of error subspace filter algorithms rendered large-scale data assimilation with Kalman-type filters possible. However, filters like the EnKF, SEEK, and SEIK algorithms still exhibit a high computational complexity. The evolution of the approximated covariance matrix still requires a vast amount of computation time, in particular for large-scale models. Also the memory requirements are large since, besides the fields required for the numerical model itself, the ensemble or mode matrix has to be allocated. In addition, several matrices need to be allocated temporarily for the analysis and resampling phases of the filter algorithms.

The computational and memory requirements can be alleviated by the use of parallel computers. Using parallelization methods like the Message Passing Interface (MPI) [27], the ensemble or mode matrix can be distributed over several processes. Thus, the memory requirements of each single process can be reduced. Additionally, the inherent parallelism of the error subspace Kalman filters (ESKF) can be exploited. The evolution of different ensemble states is independent, as was mentioned in chapter 3. Thus, the forecast phase can be parallelized by distributing the state ensemble over multiple model tasks executed concurrently by different processes. The ensemble states are then evolved concurrently by the model tasks, see e.g. [17, 74]. Most of the execution time of a filtering application is usually spent in the forecast phase, while the parts for the model initialization and the execution of the analysis and resampling phases require a significantly smaller amount of time. Thus, according to Amdahl's law, the use of independent model tasks will provide a high parallel efficiency. Hence, the time required to compute a particular data assimilation problem will strongly decrease when an increasing number of processes is used for the computations.

This is an advantage over the popular adjoint method which is inherently serial due to the alternating forward and backward evolutions with the numerical model and its adjoint, as was discussed in section 1.2. Hence, the adjoint method allows only for a decomposition of the model domain to distribute the evolutions over multiple processes. The value of the cost function and the gradient would then be gathered by a single process to update the control variables according to the chosen optimization algorithm. Trémolet and Le Dimet [82, 81] proposed to distribute also the phase in which the control variables are updated. In this case, the cost functional J is evaluated by each process on its local sub-domain. Further, the gradient of J is computed for

the local cost functional. To ensure continuity of the model fields between neighboring sub-domains, the cost functional is augmented by an additional term penalizing differences of the model fields at the boundaries of neighboring sub-domains. Thus, this difference of the boundary values is also to be minimized by the optimization algorithm. The speedup of the distributed adjoint method will not be ideal. This is due to the exchange of data between neighboring sub-domains during the evolutions as well as for the computation of the cost function. In addition, it is not assured that the minimization converges with the same number of iterations on each sub-domain.

The parallelization of filter algorithms has been discussed most extensively in the context of the EnKF algorithm [44, 45, 46, 36]. Here, different approaches have been examined. The forecast phase can either be parallelized by exploiting its inherent parallelism, or by a domain-decomposition of the model grid. The analysis phase can also be parallelized by either holding sub-ensembles of full model states on each process or by operating on full ensembles of sub-states corresponding to a sub-domain. In the context of a low-rank filter, the parallelization of the RRSQRT algorithm has been examined [70, 74, 73]. Here, the same parallelization strategies of domain-decomposition and distributed ensembles as for the EnKF algorithm have been discussed.

For the implementation of filter algorithms with existing numerical models, a clear logical separation between the filter and model parts of a data assimilation application is valuable. In addition, a well defined interface structure for the transfer of data between the filter and model parts is required. To support a separation between these two parts of a filtering application, the interface systems SESAM [75] and PALM [60] have been developed. SESAM is implemented using UNIX shell scripts which control the execution of separated program executables like the numerical model and the program computing the analysis and resampling phases of the filter. Data transfers between the programs are performed using disk files. The structure of SESAM has been developed with the aim of avoiding changes to the source code of the numerical model when using it for data assimilation. Since SESAM is based on shell scripts, it does not support multiple model tasks. The numerical efficiency of a data assimilation application implemented with SESAM will not be optimal since the disk operations used for data transfers are extremely slow compared with memory operations.

The coupler system PALM uses program subroutines which are instrumented with meta information for the PALM system. The data assimilation program is assembled using the prepared subroutines and a library of driver and algebraic routines supplied by PALM. For a filter algorithm, the resulting program supports the concurrent evaluation of multiple model tasks. In addition, a better numerical efficiency can be expected compared with SESAM, since data transfers are performed by subroutine calls. Thus, no disk operations will be required. For the implementation of a data assimilation application, PALM requires, however, to assemble the algorithm from separate subroutines. Since the numerical model is used as a subroutine, it must not be implemented with a main program. Thus, the model has to be adapted to fulfill this requirement. In addition, the control of the filtering program will emanate from the driver routine

of PALM. The numerical model is reduced to a module in the PALM system. This might lead to acceptance problems, since the major part of the source code for the data assimilation program is given by the numerical model.

In the following chapters, the application of the EnKF, SEEK and SEIK algorithms on parallel computers is studied. For the parallelization of the filter algorithms a two-step strategy is used:

First, the parallelization of the analysis and resampling phases is considered in chapter 7. These phases are independent from the model. Hence, the data transfer between the filter and model parts of the program is of no concern here. Both parallelization variants of distributed sub-ensembles and of domain-decomposed states are examined for all three filter algorithms. In addition, a localization of the analysis phase is discussed. This localization neglects observations beyond a chosen distance from a grid point of the model domain. It is shown that the localization is only relevant for the EnKF algorithm.

Subsequently, in chapter 8, the parallelization of the forecast phase is discussed. This phase is parallelized within a framework for parallel filtering which is developed in this chapter. The framework provides two levels of parallelism. The model and filter routines can be parallelized independently. Further, multiple model tasks can be executed concurrently. The number of processes for each model task and for the filter routines, as well as the number of parallel model tasks, are specified by the user of the data assimilation program. The framework defines an application program interface to assure a well defined calling structure of the filters. This permits to combine filter algorithms with existing model source codes which are not designed for data assimilation purposes. The structure of the framework amounts to attaching the filter algorithm to the model by adding subroutine calls to the model source code. The data assimilation program will be controlled by user-written routines. Thus, the required parameters can be initialized within the model source code. The framework permits to switch between filter algorithms in the same data assimilation program by the specification of a single parameter. In addition, the observation-related parts of the filter algorithms are implemented in routines separated from the core routines of the filter. This allows for a flexible handling of different observational data sets.

To assess the parallel efficiency of the filtering framework in chapter 9, it has been implemented with the finite element ocean model FEOM which has been recently developed at the Alfred Wegener Institute [12]. First, the data assimilation experiments of chapter 4 are extended to a more complex 3-dimensional test-case by performing twin experiments with an idealized model configuration of FEOM. To examine the filtering performance of the SEEK, SEIK, and EnKF algorithms, synthetic observations of the sea surface height are assimilated. Subsequently to these data assimilation experiments, the parallel efficiency of the filtering framework is examined. Then, the parallel efficiency of the analysis and resampling phases of the parallel filter algorithms is studied. The results will show, that the filtering framework developed in chapter 8 exhibits an excellent parallel efficiency. Furthermore, the framework and the filter algorithms are well suited for application to realistic large-scale data assimilation problems.

Chapter 7

Parallelization of the Filter Algorithms

7.1 Introduction

To cope with their high computational complexity, the error subspace Kalman filter algorithms share the benefit that they comprise some level of natural parallelism which can be exploited on parallel computers. The independence of the forecasts of the ensemble members has often been stressed for the EnKF [17], but it is also inherent in the SEIK filter. For the SEEK filter, the forecasts of the modes are independent if the gradient approximation is used. They are not independent if SEEK is used with the linearized model to evolve the modes. In this case, the nonlinear forecast of the state estimate is required at each time step to evaluate the linearization. If the numerical model is linear, either the modes or the columns of the state covariance matrix can be evolved independently in parallel even with the full Kalman filter. This has been utilized by Lyster et al. [52] to perform data assimilation with a linear 2-dimensional transport model for atmospheric chemical constituents using the (full-rank) linear Kalman filter. The authors compared parallelizations which either decompose the covariance matrix into columns or apply a decomposition in which only several rows of the covariance matrix are stored on a process. The latter method amounts to a decomposition of the model domain. While the forecast phase showed a rather good speedup in this study, the parallel efficiency of the analysis phase is only small. These results can be expected since the analysis phase involves global operations on the model domain. Hence, a parallelized analysis algorithm will contain a high amount of communication.

Applying the EnKF, Keppenne [44] exploited the inherent parallelism of the ensemble forecast in data assimilation with a 2-layer shallow water model. In the forecast, Keppenne distributed the ensemble members over the processes. (We will refer below to this type of distribution as "mode-decomposition".) For the analysis phase of the filter this work decomposed the model domain into sub-domains (referred to as "domain-decomposition") to allow for an analysis on a regional basis. This approach was further refined by Keppenne and Rienecker [45, 46] where the filter was applied to an ocean genereal circulation model (OGCM) in a model configuration for the Pacific basin.

Here, the model and the filter were parallelized by domain decomposition. In addition, a localized analysis is performed assimilating only observations within a certain distance from a grid point. A localized analysis has also been described by Ott et al. [58]. In this work the analysis was formulated using overlapping domains. Furthermore, only observations local to a domain are considered.

In the context of the RRSQRT filter, two parallelization approaches have been discussed. Roest and Vollebregt [70] split their data assimilation code into parts which are independently parallelized using different types of parallelism. Applying a mode decomposition in the forecast phase, they also utilize the inherent parallelism of this phase. Other operations on the covariance matrix, like a re-diagonalization analogous to the re-orthonormalization of the modes performed in the SEEK filter, are evaluated using distributed rows of the matrix. Segers and Heemink [74] compare mode and domain decomposition variants of the RRSQRT filter applied to an air pollution model. In this example both methods yield rather comparable values for the speedup. Segers and Heemink favor the domain decomposition method, based on their experience that the parallelization of the analysis part of the RRSQRT algorithm is easier for a domain decomposition than for a mode decomposition. They stress that this method requires a parallel, domain decomposed model.

In this chapter, we will examine the possibilities for the parallelization of the SEEK, EnKF, and SEIK algorithms. The variant of using the mode decomposition of the ensemble matrix in these filters is discussed in section 7.2. Subsequently in section 7.3 we examine the option to decompose the state vectors by a domain decomposition. Finally, we introduce in section 7.4 a formulation for a localized analysis which permits to assimilate observations within a certain distance from a grid point of the model domain. We focus on the analysis and resampling phases of these algorithms. The forecast phase is examined in connection with the development of a framework for parallel filtering in chapter 8. For parallelization, we use the Message Passing Interface (MPI) [27]. Some fundamental concepts of parallel computing are discussed in appendix A which also contains an introduction to MPI.

7.2 Parallelization over the Modes

For now, we consider a parallelization using mode-decomposition, i.e. the ensemble matrix $\mathbf{X}$, or the mode matrix $\mathbf{V}$, is distributed such that the process with rank p owns $k_p < N$ columns of the matrix. Thus, the local column indices $i_p = 1, \ldots, k_p$ correspond to the global indices $i = j_p, \ldots, j_p + k_p$ where $j_0 = 1$ and $j_p = 1 + \sum_{l=1}^{p} k_p$ for $p > 0$. This decomposition is displayed in figure 7.1 for $s + 1$ processes. Each column of $\mathbf{X}$ represents a full state vector. Since each process has direct access only to its k_p local state vectors, operations on $\mathbf{X}$ are distributed, too. For efficiency, as many computations as possible are performed in parallel during the analysis and resampling phase. Thus, also some operations on derived matrices, which appear in the filter algorithms, will be distributed. Some of these matrices are also stored distributed over the processes. If data from other processes is required, data exchanges are performed by calls to communication functions of the MPI library.

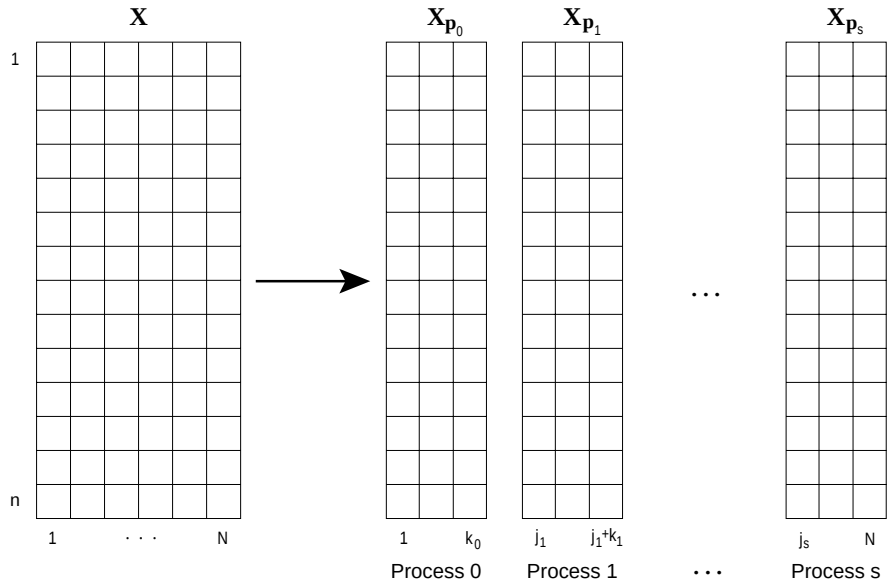


Figure 7.1: Distribution of the global ensemble matrix $\mathbf{X}$ into local sub-matrices $\mathbf{X_p}$ with mode-decomposition.

7.2.1 Distributed Operations

Using distributed matrices, we encounter in the filter algorithms several operations which have to be performed in parallel. Many of them are matrix-matrix products. If matrices were completely allocated by a single process a matrix-matrix product could be directly computed as $\mathbf{AB} = \mathbf{C}$. For distributed matrices there are, in general, three different ways of evaluating a matrix-matrix product depending on the type of distribution. These parallel matrix-matrix products are explained in table 7.1.

Other distributed operations which occur in the filter analysis and resampling phases are:

- The application of the measurement operator to the ensemble or mode matrix. E.g., in SEEK this is $\mathbf{HV}$, see equation (2.28). Only k_p columns of the matrix $\mathbf{V}$, each representing a state vector, are allocated on a process. Thus, the measurement operator is applied in a loop calling for each local column a subroutine performing the application of $\mathbf{H}$ to this column. If the full matrix $\mathbf{HV}$ is required by a single process a 'gather' operation has to be performed.

- The solution of linear equations of type $\mathbf{AB} = \mathbf{C}$. An example of this can be found in the representer formulation of the EnKF when solving equation (2.47). Here only k_p columns of the matrix $\mathbf{C}$ are allocated on a process. Thus, the solution $\mathbf{B}$ will consist of k_p local columns. If the full matrix $\mathbf{B}$ is needed by a single process, a 'gather' operation is required.

- The initialization of the observation vector $\mathbf{y}$ which has to be known by each processes. This is performed by a subroutine call. If $\mathbf{y}$ is read from a file, it is most efficient to execute the file operation only by a single process. To distribute the vector, a 'broadcast' operation is performed afterwards.

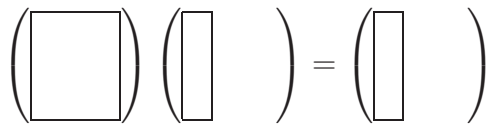
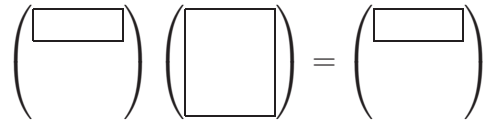
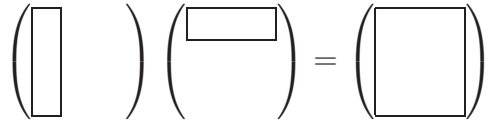
Type 1: Matrix **A** is fully allocated on each process. It is multiplied with matrix **B** from which only k_p columns are available locally. Performing the multiplication, we obtain k_p columns of the product-matrix **C**. These columns correspond to the same column indices as those available of matrix **B**. To obtain the full matrix **C** on a process a 'gather' operation has to be performed.	
Type 2: Only k_p rows of matrix **A** are available locally. This occurs, e.g., for the transpose of a column-wise distributed matrix. Matrix **B** is fully allocated on each process. The local part of the product matrix **C** consists of k_p rows whose row indices correspond to those indices of the rows of **A** which are available locally. To obtain the full matrix **C** on the local process, a 'gather' operation is required as in type 1.	
Type 3: Only k_p columns of matrix **A** and k_p rows of matrix **B** are allocated locally. The resulting product matrix **C** has the full dimension but its elements represent only a partial sum of the full matrix-matrix product. Thus, to obtain the full product **AB** on the local process, a 'reduce' operation has to be performed to sum up all partial sums distributed over the processes.	

Table 7.1: The different types of matrix-matrix products for distributed matrices. The right column sketches the differently distributed matrices.

7.2.2 SEEK

We develop the analysis algorithm of SEEK for a mode-decomposed matrix $\mathbf{V}$ such that each process will hold the updated eigenvalue matrix $\mathbf{U}^{-1}$ and the state estimate $\mathbf{x}$. This will reduce the total amount of communication, since $\mathbf{U}^{-1}$ is required by each process for the resampling phase and $\mathbf{x}$ is used by each process to compute the gradient approximation.

The parallel version of the SEEK analysis algorithm is shown as algorithm 7.1. It can be directly compared to the serial analysis algorithm 3.3. The routine is called by all processes each holding its local part $\mathbf{V_p} \in \mathbb{R}^{n \times r_p}$ of the mode matrix. In the pseudo code of the parallel algorithm the subscript p denotes an array which is private to a process. That is, the array can have a different size and hold different values on each process. Variables without this subscript are global, i.e. they have on all processes the same size and hold the same values. The application of the measurement operator on the mode matrix (lines 4-6 in algorithm 7.1) is performed only for the r_p locally allocated columns of $\mathbf{V}$. Also the subsequent product $\mathbf{R}^{-1}\mathbf{T1}$ is only computed for the local columns. The the residual $\mathbf{d}$ is initialized in lines 11 to 13 equally by all processes. This operation does, in general, require negligible computation time compared with the other operations of the analysis. Hence, initializing $\mathbf{d}$ by each process will not be problematic for the parallel efficiency. A 'broadcast' operation is hidden in the initialization of the observation vector, as was explained in the preceding section. The matrix-vector product in line 14 yields the local part of a distributed vector. Although the full vector **t3** has to be initialized by a concluding 'allgather' operation, this variant to obtain **t3** is faster than performing an 'allgather' on the much larger matrix **T2**. The following solver step (line 16) has to be performed by each process. We will see that this operation can limit the overall parallel efficiency of the SEEK analysis algorithm in mode decomposition. The final update of the state estimate is performed with the local matrix $\mathbf{V_p}$. We divide this operation into two parts. First we compute the analysis increment $\mathbf{\Delta x}$ using a matrix-matrix product of type 2 followed by an 'allreduce' operation for the analysis increment. Finally, the increment is added to the forecast state estimate $\mathbf{x}$ in order to obtain the analysis state estimate on each process. Due to the non-parallelized solver step and the required global communications, we can not expect that the mode-parallel SEEK analysis algorithm scales well.

In the resampling phase of SEEK, the mode vectors distributed over the processes are re-orthonormalized. The serial algorithm is shown as algorithm 3.7. The parallel algorithm, shown as algorithm 7.2, distributes the inversion of the matrix **Uinv**. Also the computations of the matrices **T1** and **T2** are parallelized. However, global communication is required in the algorithm to obtain the matrix $\mathbf{B}$. The most expensive communication operation will be the allgather operation of the $n \times r$ matrix $\mathbf{V}$. In contrast to this, the re-initialization of the local columns of the mode matrix $\mathbf{V}$ in line 14 is performed in a distributed matrix-matrix product of type 1 which is locally a full matrix-matrix product. Hence it is evaluated independently by all processes. The resampling algorithm also contains some operations which are performed equally by all processes: The Cholesky decomposition of $\mathbf{U}$, the computation of $\mathbf{B}$, and the singular value decomposition (SVD) of $\mathbf{B}$. We will see later that these operations, together with

Subroutine SEEK_Analysis_Mode($step$,n,r,$\mathbf{x}$,$\mathbf{Uinv}$,$\mathbf{V_p}$)
int $step$ {time step counter,input}
int n {state dimension, input}
int r {rank of covariance matrix, input}
real $\mathbf{x}(n)$ {state forecast, input/output}
real $\mathbf{Uinv}(r,r)$ {inverse eigenvalue matrix, input/output}
real $\mathbf{V_p}(n,r_p)$ {local mode matrix, input/output}
real $\mathbf{T1},\mathbf{t3},\mathbf{t4},\mathbf{d},\mathbf{y},\mathbf{\Delta x}$ {fields to be allocated}
real $\mathbf{T1_p},\mathbf{T2_p},\mathbf{t3_p},\mathbf{Uinv_p},\mathbf{\Delta x_p}$ {fields to be allocated}
int r_p {number of local columns of $\mathbf{V_p}$}
int m {dimension of observation vector}
int i {ensemble loop counter}

1: call Get_Dim_Obs($step,m$) {by each process}
2: Allocate fields: $\mathbf{T1}(m,r),\mathbf{t3}(r),\mathbf{t4}(r),\mathbf{d}(m),\mathbf{y}(m),\mathbf{\Delta x}(n)$,
3: $\mathbf{T1_p}(m,r_p),\mathbf{T2_p}(m,r_p),\mathbf{t3_p}(r_p),\mathbf{Uinv_p}(r,r_p),\mathbf{\Delta x_p}(n)$

4: **for** i=1,r_p **do**
5: call Measurement_Operator($step,n,m,\mathbf{V_p}(:,i),\mathbf{T1_p}(:,i)$) {local columns}
6: **end for**
7: allgather $\mathbf{T1}$ from $\mathbf{T1_p}$ {global MPI operation}
8: call RinvA($step,m,r,\mathbf{T1_p},\mathbf{T2_p}$) {operate only on local columns}
9: $\mathbf{Uinv_p} \leftarrow \mathbf{Uinv_p} + \mathbf{T1}^T\mathbf{T2_p}$ {matrix-matrix product type 1}
10: allgather $\mathbf{Uinv}$ from $\mathbf{Uinv_p}$ {global MPI operation}

11: call Measurement_Operator($step,n,m,\mathbf{x},\mathbf{d}$) {by each process}
12: call Measurement($step,m,\mathbf{y}$) {by each process}
13: $\mathbf{d} \leftarrow \mathbf{y} - \mathbf{d}$ {by each process}

14: $\mathbf{t3_p} \leftarrow \mathbf{T2_p}^T\mathbf{d}$ {matrix-matrix product of type 2}
15: allgather $\mathbf{t3}$ from $\mathbf{t3_p}$ {global MPI operation}
16: solve $\mathbf{Uinv}\ \mathbf{t4} = \mathbf{t3}$ for $\mathbf{t4}$ {by each Process}
17: $\mathbf{\Delta x_p} \leftarrow \mathbf{V_p}\ \mathbf{t4}$ {local state increment, matrix-vector product of type 3}
18: allreduce summation of $\mathbf{\Delta x}$ from $\mathbf{\Delta x_p}$ {global MPI operation}
19: $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{\Delta x}$ {by each process}
20: De-allocate local analysis fields

Algorithm 7.1: Structure of the parallel filter analysis routine for the SEEK algorithm. The mode matrix $\mathbf{V}$ is distributed such that each process holds r_p columns $\mathbf{V_p}$ of $\mathbf{V}$. The subscript p denotes variables which are private to a process. These can be either the locally allocated parts of distributed fields or full-size fields which hold different values on different processes.

Subroutine SEEK_Reortho_Mode(n,r,$\mathbf{Uinv}$,$\mathbf{V_p}$)
int n {state dimension, input}
int r {rank of covariance matrix, input}
real $\mathbf{Uinv}(r,r)$ {inverse eigenvalue matrix, input/output}
real $\mathbf{V_p}(n,r_p)$ {local mode matrix, input/output}
int r_p {number of local columns of $\mathbf{V_p}$}
real $\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D},\mathbf{U},\mathbf{V},\mathbf{T2}$ {fields to be allocated}
real $\mathbf{U}_p,\mathbf{I_p},\mathbf{T1_p},\mathbf{T2_p},\mathbf{T3_p},\mathbf{T4_p}$ {fields to be allocated}

1: Allocate fields: $\mathbf{A}(r,r),\mathbf{B}(r,r),\mathbf{C}(r,r),\mathbf{D}(r,r),\mathbf{U}(r,r),\mathbf{V}(n,r)$,
2: $\mathbf{T2}(r,r),\mathbf{U_p}(r,r_p),\mathbf{I_p}(r,r_p),\mathbf{T1_p}(r,r_p),\mathbf{T2_p}(r,r_p),\mathbf{T3_p}(r,r_p),\mathbf{T4_p}(r,r_p)$

3: $\mathbf{I_p} \leftarrow \mathbf{I}(:, j_p : j_p + r_p - 1)$ {local columns of identity matrix}
4: Solve $\mathbf{Uinv}\ \mathbf{U_p} = \mathbf{I_p}$ for $\mathbf{U_p}$ {get local columns of $\mathbf{U}$}
5: allgather $\mathbf{U}$ from $\mathbf{U_p}$ {global MPI operation}
6: Cholesky decomposition: $\mathbf{U} = \mathbf{A}\mathbf{A}^T$ {by each process}
7: allgather $\mathbf{V}$ from $\mathbf{V_p}$ {global MPI operation}
8: $\mathbf{T1_p} \leftarrow \mathbf{V}^T\ \mathbf{V_p}$ {matrix-matrix product of type 1}
9: $\mathbf{T2_p} \leftarrow \mathbf{A}^T\ \mathbf{T1_p}$ {matrix-matrix product of type 1}
10: allgather $\mathbf{T2}$ from $\mathbf{T2_p}$ {global MPI operation}
11: $\mathbf{B} \leftarrow \mathbf{T2}\ \mathbf{A}$ {by each process}

12: SVD: $\mathbf{B} = \mathbf{C}\ \mathbf{D}\ \mathbf{C}^T$ {by each process}
13: $\mathbf{T3_p} \leftarrow \mathbf{C}\ \mathbf{D}(:, j_p : j_p + r_p - 1)^{-1/2}$ {Initialize $\mathbf{T3_p}$ using local columns of $\mathbf{D}$}
14: $\mathbf{T4_p} \leftarrow \mathbf{A}\ \mathbf{T3_p}$ {matrix-matrix product of type 1}
15: $\mathbf{V_p} \leftarrow \mathbf{V}\ \mathbf{T4_p}$ {matrix-matrix product of type 1}
16: $\mathbf{Uinv} \leftarrow \mathbf{D}^{-1}$ {by each process}
17: De-allocate local analysis fields

Algorithm 7.2: Structure of the parallel version of the re-orthonormalization routine for the SEEK algorithm. Matrix $\mathbf{D}$ holding the singular values of $\mathbf{T3}$ is introduced here for clarity. In the program, it is allocated as a vector holding the eigenvalues of $\mathbf{T3}$. The large number of matrices of sizes $r \times r$ or $r \times r_p$ is introduced in the pseudo code for clarity. In the program itself, only two matrices of size $r \times r_p$ and three of size $r \times r$ are allocated. The index j_p denotes the index of the first column of $\mathbf{V_p}$ in the global matrix $\mathbf{V}$.

the required communications, will limit the overall parallel efficiency of the algorithm. An obvious drawback of the presented algorithm is that the full matrix $\mathbf{V}$ has to be allocated on each process. It is, however, possible to formulate the algorithm with a block structure allocating only several rows of $\mathbf{V}$ at a time. This will involve a lot of communication operations of smaller amounts of data. The total amount of communicated data will be twice as large since the full information on $\mathbf{V}$ is required for the operations in line 7 and in line 14.

7.2.3 EnKF

The parallel analysis algorithm for the EnKF with a mode-decomposed ensemble matrix $\mathbf{X}$ is shown as algorithm 7.3. The serial algorithm has been given as algorithm 3.5.

The routine is called by all processes each holding its local part $\mathbf{X_p} \in \mathbb{R}^{n \times N_p}$ of the ensemble matrix. In the parallel algorithm, the computation of the mean of the ensemble projected onto observation space in line 7 corresponds to a matrix-matrix product of type 3 in which the second matrix has only one column whose entries are equal to N^{-1}. An allreduce summation is necessary to obtain the ensemble mean on all processes. This is analogous for the computation of the ensemble mean state in line 22. The full matrix **T1** is initialized by each process using an allgather operation in line 12. Subsequently, the computation of **T3** is performed equally by all processes. Alternatively, several columns of **T3** could be computed first via a matrix-matrix product of type 1. Then the full matrix **T3** would be initialized by all processes by an allgather operation. Whether this parallelized variant is faster than computing **T3** directly by each process will depend on the ratio of computation to communication performance.

In the EnKF, an ensemble of residuals has to be computed from an ensemble of observations. The observations are generated in the subroutine *Enkf_Obs_Ensemble* which will involve a broadcast operation if the observation vector is read from a file. The computation of the local residual ensemble $\mathbf{D_p}$ itself (lines 15 to 19) is performed independently by each process.

The solver step for the influence amplitudes $\mathbf{B}$ in line 20 is distributed over the processes. Thus, local amplitudes $\mathbf{B_p}$ are computed using the LAPACK routine *DGESV*. The parallel efficiency of this operation is, however, limited since the LU-decomposition of $\mathbf{T3} \in \mathbb{R}^{m \times m}$ is performed by each process. The final update of the local state ensemble $\mathbf{X_p}$ in line 28 is performed independently by each process. The preparations for the update, which are performed from lines 22 to 27, include the initializations of the ensemble mean $\mathbf{x}$ and the matrix **T5** by communication operations. To avoid the allocations of the matrices $\mathbf{T5_p}$ and **T5** as well as those of the vectors $\mathbf{x_p}$ and $\mathbf{x}$, we use a block formulation for lines 22 to 28.

In the mode-decomposed EnKF analysis algorithm, the computation of **T3** is not parallelized. In addition, the solver step for the representer amplitudes can not be expected to show a good parallel efficiency. Next to these operations, several global communication operations have to be performed. These properties of the mode-decomposed algorithm will limit the parallel efficiency.

In the mode-decomposed EnKF algorithm, the global matrix $\mathbf{T3} \in \mathbb{R}^{m \times m}$ is computed by each process since it is required for the solver step in line 20. This requirement presents a particular issue for the mode-decomposed EnKF filter. Next to the requirement to allocate this matrix, the operations involving **T3** will be costly. To reduce the operational complexity, it is possible to sequentially assimilate batches of independent observations. This technique has been discussed in section 3.4. Indeed, it will reduce the effective dimension of the observation vector. Accordingly, the memory requirements are reduced. Furthermore, the number of operations is decreased, since the complexity of the matrix-matrix product in line 13 scales with $\mathcal{O}(m^2)$ and that of the solver step in line 20 is $\mathcal{O}(m^3 + m^2 N)$.

Subroutine EnKF_Analysis_Mode($step$,n,N_p,$\mathbf{X_p}$)
int $step$ {time step counter,input}
int n {state dimension, input}
int N {ensemble size, input}
real $\mathbf{X_p}(n, N_p)$ {local ensemble matrix, input/output}
real $\mathbf{T1}, \mathbf{t2}, \mathbf{T3}, \mathbf{T5}, \mathbf{x}$ {fields to be allocated}
real $\mathbf{T1_p}, \mathbf{t2_p}, \mathbf{t4_p}, \mathbf{T5_p}, \mathbf{T6_p}, \mathbf{D_p}, \mathbf{B_p}, \mathbf{x_p}$ {fields to be allocated}
int N_p {local ensemble size}
int m {dimension of observation vector}
int i {ensemble loop counter}

1: call Get_Dim_Obs($step, m$) {by each process}
2: Allocate fields: $\mathbf{T1}(m, N), \mathbf{t2}(m), \mathbf{T3}(m, m), \mathbf{T5}(n, N), \mathbf{x}(n), \mathbf{T1_p}(m, N_p)$,
3: $\mathbf{t2_p}(m), \mathbf{t4_p}(m), \mathbf{T5_p}(n, N_p), \mathbf{T6_p}(N, N_p), \mathbf{B_p}(m, N_p), \mathbf{D_p}(m, N_p), \mathbf{x_p}(n)$

4: **for** i=1,N_p **do**
5: call Measurement_Operator($step, n, m, \mathbf{X_p}(:, i), \mathbf{T1_p}(:, i)$) {local columns}
6: **end for**
7: $\mathbf{t2_p} \leftarrow N^{-1} \sum_{i=1}^{N_p} \mathbf{T1_p}(:, i)$ {local mean of projected ensemble}
8: allreduce summation of $\mathbf{t2}$ from $\mathbf{t2_p}$ {global MPI operation}
9: **for** i=1,N_p **do**
10: $\mathbf{T1_p}(:, i) \leftarrow \mathbf{T1_p}(:, i) - \mathbf{t2}$ {local columns}
11: **end for**
12: allgather $\mathbf{T1}$ from $\mathbf{T1_p}$ {global MPI operation}
13: $\mathbf{T3} \leftarrow (N-1)^{-1}\ \mathbf{T1}\ \mathbf{T1}^T$ {by each process}
14: call RplusA($step$,m,$\mathbf{T3}$) {by each process}

15: call Enkf_Obs_Ensemble($step$,m,N_p,$\mathbf{D_p}$) {get local ensemble of observations}
16: **for** i=1,N_p **do**
17: call Measurement_Operator($step, n, m, \mathbf{X}_p(:, i), \mathbf{t4_p}$) {local columns}
18: $\mathbf{D}_p(:, i) \leftarrow \mathbf{D_p}(:, i) - \mathbf{t4_p}$ {local ensemble of residuals}
19: **end for**

20: solve $\mathbf{T3}\ \mathbf{B_p} = \mathbf{D_p}$ for $\mathbf{B_p}$ {get local representer amplitudes}
21: $\mathbf{T6_p} \leftarrow \mathbf{T1}^T\ \mathbf{B_p}$ {matrix-matrix product of type 1}
22: $\mathbf{x_p} \leftarrow N^{-1} \sum_{i=1}^{N_p} \mathbf{X_p}(:, i)$ {local ensemble mean state}
23: allreduce summation of $\mathbf{x}$ from $\mathbf{x_p}$ {global MPI operation}
24: **for** i=1,N_p **do**
25: $\mathbf{T5_p}(:, i) \leftarrow \mathbf{X_p}(:, i) - \mathbf{x}$ {local columns}
26: **end for**
27: allgather $\mathbf{T5}$ from $\mathbf{T5_p}$ {global MPI operation}
28: $\mathbf{X_p} \leftarrow \mathbf{X_p} + (N-1)^{-1}\ \mathbf{T5}\ \mathbf{T6_p}$ {matrix-matrix product of type 1}
29: De-allocate local analysis fields

Algorithm 7.3: Structure of the parallel filter analysis routine for the EnKF algorithm using the representer update variant for a non-singular matrix $\mathbf{T5}$. Matrix $\mathbf{B_p}$ is not allocated individually but stored in $\mathbf{D_p}$. Analogously, $\mathbf{t4}$ is stored in $\mathbf{t2}$. The allocation of the full array $\mathbf{T5}$ can be avoided by a block formulation for line 28.

7.2.4 SEIK

The analysis algorithm of the SEIK filter is very similar to that of the SEEK filter. Hence, also the parallelization is almost identical in both cases. Discussing the parallelization of SEIK, we focus on the unique parts of it. The parallel SEIK analysis algorithm is shown as algorithm 7.4 while the serial analysis has been shown as algorithm 3.4.

An additional operation in the analysis algorithm of SEIK compared with SEEK is the matrix-matrix product in line 7. Here the ensemble matrix projected onto the observation space (**T1** in the pseudo code) is multiplied with matrix **T** defined by equation (2.62). As has been discussed in section 3.3, this operation is most efficiently implemented taking into account the particular choice of **T**. Accordingly, this multiplication involves the subtraction of the global ensemble mean of **T1** from each column of this matrix. This mean is computed as the means in the EnKF, i.e. by calculating local means followed by an allreduce summation. The computed ensemble mean is subtracted from each of the local ensemble states. In line 21, the product **T t5** is computed. Following the discussion in section 3.3, the mean value of the elements of **t5** is computed and subsequently subtracted from each column. The final column is initialized by the negative of the mean value. The product **T t5** does not require communication, since **t5** is allocated on each process. Other additional operations in the analysis phase of SEIK are the computation of the ensemble mean in line 13, which is computed as in the EnKF, and the initialization of matrix **G** in line 10. This operation is parallelized by initializing only r_p local columns. These are required for the subsequent computation of **Uinv** which is a matrix-matrix product of type 1 followed by an allgather operation. Since the solver step in line 21 is not parallelized and several global communication operations are performed, we cannot expect that the mode-parallel SEIK analysis algorithm scales perfectly.

A particular parallelization issue of the SEIK filter is that matrix **T2** consists of only r columns, while **T1** contains $N = r + 1$ columns. Hence, for the load-balancing of the analysis algorithm the application of **T** is problematic. Since the forecast phase usually requires the most computation time, we chose a configuration in which each process holds the same number $N_p = k$ of ensemble states (I.e. the same number of columns in the local matrices $\mathbf{X_p}$ and $\mathbf{T1_p}$). Computing the product $\mathbf{T1_p}$ **T** reduces the number of overall columns by one. Accordingly, one of the processes (usually that one with the highest rank) holds only $k - 1$ local columns of $\mathbf{T1_p}$ **T**, while all other processes hold k local columns. Due to this, one of the processes executes less operations than the other processes and will complete work earlier. However, this is inevitable if the ensemble has to be distributed evenly in order to obtain the best speed up in the forecast phase. For the parallel algorithm, this has no special implications, as long as the number of columns in matrix $\mathbf{T2_p}$ is not reduced to zero on one of the processes.

In the resampling algorithm of SEIK, a new ensemble of states is computed on the basis of the forecasted state ensemble **X**. The parallel algorithm is shown as algorithm 7.5. It can be compared with the serial algorithm 3.8. The Cholesky decomposition in line 2 is performed equally by all processes. The solver step for the local

Subroutine SEIK_Analysis_Mode($step$,n,N,$\mathbf{x}$,$\mathbf{Uinv}$,$\mathbf{X_p}$)
int $step$ {time step counter,input}
int n {state dimension, input}
int N {ensemble size, input}
real $\mathbf{x}(n)$ {local state estimate, output}
real $\mathbf{Uinv}(r,r)$ {inverse eigenvalue matrix, output}
real $\mathbf{X_p}(n,N_p)$ {local ensemble matrix, input/output}
real $\mathbf{T2},\mathbf{t4},\mathbf{t5},\mathbf{t6},\mathbf{y},\mathbf{d},\mathbf{\Delta x}$, {fields to be allocated}
real $\mathbf{T1_p},\mathbf{T2_p},\mathbf{T3_p},\mathbf{t4_p},\mathbf{G_p},\mathbf{Uinv_p},\mathbf{x_p},\mathbf{\Delta x_p}$ {fields to be allocated}
int r {rank of covariance matrix, $r=N-1$}
int r_p {number of local columns of covariance matrix}
int N_p {local ensemble size}
int m {dimension of observation vector}
int i {ensemble loop counter}

1: call Get_Dim_Obs($step,m$) {by each process}
2: Allocate fields: $\mathbf{T2}(m,r),\mathbf{t4}(r),\mathbf{t5}(r),\mathbf{t6}(N),\mathbf{y}(m),\mathbf{d}(m),\mathbf{\Delta x}(n),\mathbf{T1_p}(m,N_p)$,
3: $\mathbf{T2_p}(m,r_p),\mathbf{T3_p}(m,r_p),\mathbf{t4_p}(r_p),\mathbf{G_p}(r,r_p),\mathbf{Uinv_p}(r,r_p),\mathbf{x_p}(n),\mathbf{\Delta x_p}(n)$

4: **for** i=1,N_p **do**
5: call Measurement_Operator($step,n,m,\mathbf{X_p}(:,i),\mathbf{T1_p}(:,i)$) {user supplied}
6: **end for**
7: $\mathbf{T2_p} \leftarrow \mathbf{T1_p}\ \mathbf{T}$ {implemented with $\mathbf{T}$ as operator}
8: allgather $\mathbf{T2}$ from $\mathbf{T2_p}$ {global MPI operation}
9: call RinvA($step,m,r,\mathbf{T2_p},\mathbf{T3_p}$) {operate only on local columns}
10: $\mathbf{G_p} \leftarrow (N^{-1}(\mathbf{T}^T\ \mathbf{T})^{-1})_\mathbf{p}$ {implemented as direct initialization}
11: $\mathbf{Uinv_p} \leftarrow \mathbf{G_p} + \mathbf{T2}^T\mathbf{T3_p}$ {matrix-matrix product of type 1}
12: allgather $\mathbf{Uinv}$ from $\mathbf{Uinv_p}$ {global MPI operation}

13: $\mathbf{x_p} \leftarrow N^{-1}\sum_{i=1}^{N_p}\mathbf{X_p}(:,i)$ {get local ensemble mean state}
14: allreduce summation of $\mathbf{x}$ from $\mathbf{x_p}$ {global MPI operation}
15: call Measurement_Operator($step,n,m,\mathbf{x},\mathbf{d}$) {user supplied}
16: call Measurement($step,m,\mathbf{y}$) {user supplied}
17: $\mathbf{d} \leftarrow \mathbf{y} - \mathbf{d}$

18: $\mathbf{t4_p} \leftarrow \mathbf{T3_p}^T\mathbf{d}$ {matrix-matrix product of type 2}
19: allgather $\mathbf{t4}$ from $\mathbf{t4_p}$ {global MPI operation}
20: solve $\mathbf{Uinv\ t5} = \mathbf{t4}$ for $\mathbf{t5}$ {by each process}
21: $\mathbf{t6} \leftarrow \mathbf{T}\ \mathbf{t5}$ {implemented with $\mathbf{T}$ as operator}
22: $\mathbf{\Delta x_p} \leftarrow \mathbf{X_p}\ \mathbf{t6}(j_p : j_p + N_p - 1)$ {local increment, mat.-vec. product of type 3}
23: allreduce summation of $\mathbf{\Delta x}$ from $\mathbf{\Delta x_p}$ {global MPI operation}
24: $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{\Delta x}$ {by each process}
25: De-allocate local analysis fields

Algorithm 7.4: Structure of the parallel filter analysis routine for the SEIK algorithm. The arrays $\mathbf{T2_p}$ and $\mathbf{t5}$ are introduced for clarity. Their contents are stored respectively in $\mathbf{T1_p}$ and $\mathbf{t4}$. The index j_p denotes the index of the first column of $\mathbf{X_p}$ in $\mathbf{X}$.

Subroutine SEIK_Resample_Mode(n,N,$\mathbf{x}$,$\mathbf{Uinv}$,$\mathbf{X_p}$)
int n {state dimension, input}
int N {ensemble size, input}
real $\mathbf{x}(n)$ {state analysis vector, input}
real $\mathbf{Uinv}(r,r)$ {inverse eigenvalue matrix, input}
real $\mathbf{X_p}(n,N_p)$ {ensemble matrix, input/output}
real $\mathbf{T1}, \mathbf{T2_p}, \mathbf{C}, \mathbf{\Omega_p}^T, \mathbf{X}$ {fields to be allocated}
int r {rank of covariance matrix, $r = N-1$}
int N_p {local ensemble size}

1: Allocate fields: $\mathbf{T1}(r,N), \mathbf{T2_p}(N,N_p), \mathbf{C}(r,r), \mathbf{\Omega_p}^T(r,N_p), \mathbf{X}(n,N)$

2: Cholesky decomposition: $\mathbf{Uinv} = \mathbf{C}\ \mathbf{C}^T$ {by each process}
3: initialize $\mathbf{\Omega_p}^T$ {local columns}
4: solve $\mathbf{C}^T\mathbf{T1_p} = \mathbf{\Omega_p}^T$ for $\mathbf{T1_p}$ {local columns}
5: $\mathbf{T2_p} \leftarrow \mathbf{T}\ \mathbf{T1_p}$ {implemented with $\mathbf{T}$ as operator}
6: allgather $\mathbf{X}$ from $\mathbf{X_p}$ {global MPI operation}
7: **for** i=1,N_p **do**
8: $\mathbf{X_p}(:,i) \leftarrow \mathbf{x}$
9: **end for**
10: $\mathbf{X_p} \leftarrow \mathbf{X_p} + N^{1/2}\ \mathbf{X}\ \mathbf{T2_p}$ {matrix-matrix product of type 1 with DGEMM}
11: De-allocate local analysis fields

Algorithm 7.5: Structure of the parallel resampling routine for the SEIK algorithm. The matrix $\mathbf{T1_p}$ is not allocated in the program. Its contents are stored in $\mathbf{\Omega}^T$. To avoid the allocation of $\mathbf{X}$, lines 6 to 10 can be implemented in block formulation.

columns of $\mathbf{T1}$ in line 4 and the product $\mathbf{T}\ \mathbf{T1_p}$ (line 5) are parallelized. The latter operation is implemented as in the analysis algorithm. The initialization of the new ensemble matrix in line 10 is executed in parallel, too. Since this operation requires the information on all ensemble members in $\mathbf{X} \in \mathbb{R}^{n \times N}$, this matrix is initialized by all processes by an allgather operation (line 6). This operation will be very costly due to the large dimension of $\mathbf{X}$. To avoid the requirement to store the full matrix $\mathbf{X}$, we use a block formulation for the resampling. Therefore a loop is built around lines 5 to 10. In each cycle of this loop, only a couple of rows of the global matrix $\mathbf{X}$ are allocated and gathered at a time. In line 10 only the corresponding rows of $\mathbf{X_p}$ are updated.

7.2.5 Comparison of Communication and Memory Requirements

For comparison of the communication requirements of the three filter algorithms, table 7.2 summarizes the sizes of the arrays involved in MPI operations.

The amount of communicated data in the mode-parallel analysis algorithm of SEIK is larger than for SEEK. This is caused by the product $\mathbf{T1_p}\ \mathbf{T}$ in line 7 of algorithm 7.4 and the computation of the ensemble mean in line 14. In the resampling algorithm

Table 7.2: Sizes of arrays involved in global MPI operations in the analysis and resampling phases of the SEEK and SEEK algorithms and in the analysis phase of the EnKF algorithm. Next to the matrix size, the name of the matrix is given as well as the information whether the MPI operation is an allgather (g) or allreduce (r) operation.

	EnKF	SEEK	SEIK
analysis	mN (**T1**, g) nN (**T5**, g) m (**t2**, r) n (**x**, r)	mr (**T1**, g) r^2 (**Uinv**, g) r (**t3**, g) n ($\mathbf{\Delta x}$, r)	mr (**T2**, g) r^2 (**Uinv**, g) r (**t4**, g) n (**x**, r) n ($\mathbf{\Delta x}$, r) m (in $\mathbf{T1_pT}$, r)
re-sampling		r^2 (**U**, g) nr (**V**, g) r^2 (**T2**, g)	nN (**X**, g)

of SEEK, the global mode matrix $\mathbf{V} \in \mathbb{R}^{n \times r}$ has to be initialized by all processes using an allgather operation. Analogously the ensemble matrix $\mathbf{X} \in \mathbb{R}^{n \times N}$ has to be initialized in resampling algorithm of SEIK. In the resampling algorithm of SEEK, also the much smaller matrices **U** and **T2** are gathered.

The communication requirements of the EnKF algorithm are similar to those of the SEEK and SEIK algorithms. In the EnKF, the ensemble update is computed within the analysis, while SEEK and SEIK have additional resampling routines. Due to this, the EnKF includes the allgather operation on the matrix $\mathbf{T5} \in \mathbb{R}^{n \times N}$ which is the analogue to the allgather operations of **V** or **X** performed respectively in the resampling phases of SEEK and SEIK.

Concerning memory requirements, the mode-decomposition only permits to distribute some fields which hold ensemble quantities. Other arrays, which hold ensembles of observation-related vectors like **T1** in SEEK and EnKF, are not decomposed. Thus, the scalability of the memory requirements is limited. Next to these non-distributed arrays, additional private arrays have to be allocated. Some of these, like $\mathbf{T2_p} \in \mathbb{R}^{m \times r_p}$ in algorithm 7.1, involve the observation dimension. These arrays increase the overall memory requirements. Other arrays which involve the state dimension n, are less problematic. Using block formulations, it is not necessary to allocate these arrays in their full size. A particular memory issue is the allocation of the full mode matrix $\mathbf{V} \in \mathbb{R}^{n \times r}$ in the resampling algorithm of SEEK. As has been discussed in section 7.2.2, the allocation of this very large array can only be avoided by a block formulation. This will, however, require to gather the full information on **V** twice. In the case of the EnKF algorithm, the allocation of the matrix $\mathbf{T3} \in \mathbb{R}^{m \times m}$ is required. If very large data sets have to be assimilated, this memory requirement can be problematic. In this case, the sequential assimilation of independent observation batches with smaller dimension m will reduce the memory requirements.

7.3 Filtering with Domain Decomposition

In the case of domain-decomposition, the ensemble matrix $\mathbf{X}$, or the mode matrix $\mathbf{V}$, is distributed such that the process with rank p holds $k_p < n$ rows of the matrix. The distribution of the ensemble matrix is sketched in figure 7.2. The local row indices $i_p = 1, \ldots, k_p$ of the matrix owned by process p correspond to the global row indices $i = j_p, \ldots, j_p + k_p$ where $j_0 = 1$ and $j_p = 1 + \sum_{l=1}^{p} k_p$ for $p > 0$. Since each column of $\mathbf{X}$ represents a full state vector, each process now holds a part of each ensemble state. This configuration arises naturally, when the domain of a model is decomposed into several sub-domains each being located on a different process. Domain decomposition is a frequently used strategy in parallel computing [22]. If data assimilation is performed using a domain-decomposed model, it appears to be obvious to use a parallelization of the filter which follows the parallelization of the model it is applied to. This avoids possible reordering requirements of the state vectors and model fields in the communication between filter and model.

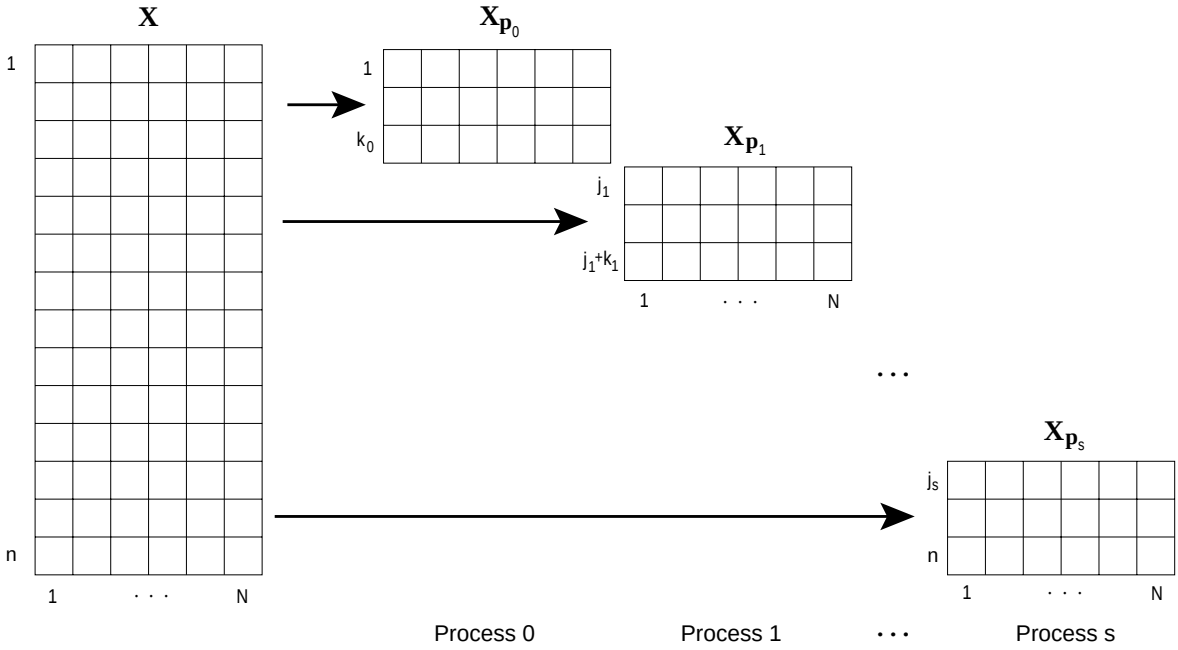


Figure 7.2: Distribution of the global ensemble matrix $\mathbf{X}$ into local sub-matrices $\mathbf{X}_\mathbf{p}$ for domain-decomposition.

As the model state is decomposed into sub-domains, also the observations should be domain-decomposed. This allows for a better parallel efficiency of the filter analysis algorithms. If the observations are distributed rather evenly in space, the decomposition of the observations should follow that of the model state. However, the decomposition of the observation vector does not need to follow that of the model state. This provides the freedom to choose a decomposition which yields an even distribution of the observation vector over the processes. This can be important for the load-balancing of the filter analysis algorithm if the observations are irregularly distributed in space.

7.3.1 Distributed Operations

Using domain decomposed ensemble matrices, the filter algorithms will again require distributed matrix-matrix products. As for mode-decomposition, these are of the types described in table 7.1.

Other distributed operations occurring in the filter analysis and resampling algorithms are:

- The initialization of the dimension of the observation vector which is performed in subroutine *Get_Dim_Obs*. If the observation space is decomposed into sub-domains, the call to *Get_Dim_Obs* has to provide the size of the local sub-domain of the observation space.
- The application of the measurement operator $\mathbf{H}$ to a state vector or the ensemble or mode matrix. In contrast to the mode-decomposition discussed above, each process holds information on all ensemble members contained in the ensemble or mode matrix, but only the about the local sub-domain. Due to this, the application of the observation operator may require communications of data, e.g. if interpolations are performed which require state information from adjacent sub-domains. Communication operations will be also necessary if the domain-decompositions of the observations and the model state are different.
- The initialization of the observation vector $\mathbf{y}$. The call to the subroutine *Measurement* has to initialize the part of the observation vector which lies in the local sub-domain of the distributed observation space. If the observation vector is read from a file, the file operation should be performed only by a single process. Thus, the initialization of $\mathbf{y}$ will involve communication operations to distribute the observation sub-vectors to other processes.
- The product of the inverse of the observation error covariance matrix $\mathbf{R}$ with the ensemble matrix projected onto observation space. This operation is performed in SEEK and SEIK by the subroutine *RinvA*. If $\mathbf{R}$ is not diagonal, the values of all elements of the state vectors in observation space are required by each process to compute the matrix-matrix product. Thus, global communication of data is necessary.

7.3.2 SEEK

The analysis algorithm of SEEK for a domain-decomposed state and mode-matrix is shown as algorithm 7.6. As has been explained above, the application of the measurement operator in lines 5 and 11, as well as the subroutine *RinvA*, can involve communication operations. In contrast to the mode-decomposed SEEK filter, no global communication operations on the ensemble matrix itself are required in the case of domain-decomposition. Only two allreduce summations on typically rather small arrays are necessary. These are allreduce summations to initialize the increment matrix $\mathbf{\Delta Uinv} \in \mathbb{R}^{r \times r}$ and to initialize the vector $\mathbf{t3} \in \mathbb{R}^r$. Matrix **Uinv** is updated equally by all processes by adding the increment matrix $\mathbf{\Delta Uinv}$. Also the solver step

Subroutine SEEK_Analysis_Domain($step$,n_p,r,$\mathbf{x_p}$,$\mathbf{Uinv}$,$\mathbf{V_p}$)
int $step$ {time step counter,input}
int n_p {state dimension on local domain, input}
int r {rank of covariance matrix, input}
real $\mathbf{x_p}(n_p)$ {local state forecast, input/output}
real $\mathbf{Uinv}(r,r)$ {inverse eigenvalue matrix, input/output}
real $\mathbf{V_p}(n_p,r)$ {local mode matrix, input/output}
real $\mathbf{t3}, \mathbf{t4}, \mathbf{\Delta Uinv}, \mathbf{d_p}, \mathbf{y_p}, \mathbf{T1_p}, \mathbf{T2_p}, \mathbf{t3_p}, \mathbf{\Delta Uinv_p}$ {fields to be allocated}
int m_p {dimension of local observation vector}
int i {ensemble loop counter}

1: call Get_Dim_Obs($step, m_p$) {get dimension for local domain}
2: Allocate fields: $\mathbf{t3}(r), \mathbf{t4}(r), \mathbf{\Delta Uinv}(r,r), \mathbf{d_p}(m_p), \mathbf{y_p}(m_p)$,
3: $\mathbf{T1_p}(m_p,r), \mathbf{T2_p}(m_p,r), \mathbf{t3_p}(r), \mathbf{\Delta Uinv_p}(r,r)$

4: **for** i=1,r **do**
5: call Measurement_Operator($step, n_p, m_p, \mathbf{V_p}(:,i), \mathbf{T1_p}(:,i)$) {local domain}
6: **end for**
7: call RinvA($step, m_p, r, \mathbf{T1_p}, \mathbf{T2_p}$) {operate only on local domain}
8: $\mathbf{\Delta Uinv_p} \leftarrow \mathbf{T1_p}^T \mathbf{T2_p}$ {matrix-matrix product type 3}
9: allreduce summation of $\mathbf{\Delta Uinv}$ from $\mathbf{\Delta Uinv_p}$ {global MPI operation}
10: $\mathbf{Uinv} \leftarrow \mathbf{Uinv} + \mathbf{\Delta Uinv}$ {by each process}

11: call Measurement_Operator($step, n_p, m_p, \mathbf{x_p}, \mathbf{d_p}$) {project local state}
12: call Measurement($step, m_p, \mathbf{y_p}$) {get local observation vector}
13: $\mathbf{d_p} \leftarrow \mathbf{y_p} - \mathbf{d_p}$ {residual for local domain}

14: $\mathbf{t3_p} \leftarrow \mathbf{T2_p}^T \mathbf{d_p}$ {matrix-matrix product of type 3}
15: allreduce summation of $\mathbf{t3}$ from $\mathbf{t3_p}$ {global MPI operation}
16: solve $\mathbf{Uinv}\ \mathbf{t4} = \mathbf{t3}$ for $\mathbf{t4}$ {by each process}
17: $\mathbf{x_p} \leftarrow \mathbf{x_p} + \mathbf{V_p}\ \mathbf{t4}$ {matrix-vector product of type 2}
18: De-allocate local analysis fields

Algorithm 7.6: Structure of the parallel SEEK analysis routine for domain decomposed states. The mode matrix $\mathbf{V}$ and the state vector $\mathbf{x}$ are distributed such that each process holds a sub-domain of dimension n_p. Also the observation space is decomposed. Thus, the observation vector $\mathbf{y}$ is distributed with each process holding a sub-domain of dimension m_p.

Subroutine SEEK_Reortho_Domain(n_p,r,$\mathbf{Uinv}$,$\mathbf{V_p}$)
int n_p {state dimension on local domain, input}
int r {rank of covariance matrix, input}
real $\mathbf{Uinv}(r,r)$ {inverse eigenvalue matrix, input/output}
real $\mathbf{V_p}(n_p,r)$ {local mode matrix, input/output}
real $\mathbf{T1},\mathbf{T2},\mathbf{T3},\mathbf{T4},\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D},\mathbf{L_p},\mathbf{U},\mathbf{T1_p}$ {fields to be allocated}

1: Allocate fields: $\mathbf{T1}(r,r),\mathbf{T2}(r,r),\mathbf{T3}(r,r),\mathbf{T4}(r,r),\mathbf{A}(r,r),\mathbf{B}(r,r),$
2: $\mathbf{C}(r,r),\mathbf{D}(r,r),\mathbf{U}(r,r),\mathbf{L_p}(n_p,r),\mathbf{T1_p}(r,r)$

3: Solve $\mathbf{Uinv}\ \mathbf{U}=\mathbf{I}$ for $\mathbf{U}$ {by each process}
4: Cholesky decomposition: $\mathbf{U}=\mathbf{A}\mathbf{A}^T$ {by each process}
5: $\mathbf{T1_p} \leftarrow \mathbf{V_p}^T\mathbf{V_p}$ {matrix-matrix product of type 3}
6: allreduce summation of $\mathbf{T1}$ from $\mathbf{T1_p}$ {global MPI operation}
7: $\mathbf{T2} \leftarrow \mathbf{T1}\ \mathbf{A}$ {by each process}
8: $\mathbf{B} \leftarrow \mathbf{A}^T\ \mathbf{T2}$ {by each process}

9: SVD: $\mathbf{T1}=\mathbf{C}\ \mathbf{D}\ \mathbf{C}^T$ {by each process}
10: $\mathbf{T3} \leftarrow \mathbf{C}\ \mathbf{D}^{-1/2}$ {by each process}
11: $\mathbf{T4} \leftarrow \mathbf{A}\ \mathbf{T3}$ {by each process}
12: $\mathbf{L_p} \leftarrow \mathbf{V_p}$
13: $\mathbf{V_p} \leftarrow \mathbf{L_p}\ \mathbf{T4}$ {matrix-matrix product of type 2}
14: $\mathbf{Uinv} \leftarrow \mathbf{D}^{-1}$ {by each process}
15: De-allocate local analysis fields

Algorithm 7.7: Structure of the parallel version of the re-orthonormalization routine for the SEEK algorithm for domain decomposed states. The matrix $\mathbf{D}$ holding the singular values of $\mathbf{B}$ is introduced here for clarity. In the program, it is allocated as a vector holding the eigenvalues of $\mathbf{B}$. Only three matrices of size $r \times r$ need to be allocated in the program. The other matrices of this size are only introduced in the pseudo code for clarity.

in line 16 is performed by all processes, as in the case of mode-decomposition. Since this operation involves the inversion of $\mathbf{Uinv}$ it can be rather costly. Over all, the domain-decomposed SEEK analysis algorithm involves less communications of data than the mode-decomposed SEEK analysis. Also less operations are executed equally by each process. Thus, we can expect that the domain-decomposed SEEK analysis will show a better parallel efficiency than the mode-decomposed analysis. The parallel efficiency will of course not be optimal due to the global communication operations and the operations which are not parallelized.

The SEEK resampling routine for a parallelization using domain decomposition is shown as algorithm 7.7. Here only the operations on matrices which involve the high dimension n are parallelized. These are the matrix-matrix product $\mathbf{V_p}^T\mathbf{V_p}$ in line 5 and the initialization of the new mode matrix $\mathbf{V_p}$ in lines 12 and 13. An allreduce summation is required to fully initialize the global matrix $\mathbf{T1}$. This operation is the

only global MPI communication which is necessary in the domain-decomposed SEEK resampling algorithm. The parts of the resampling algorithm which act on matrices of size $r \times r$ are executed equally by all processes. This can, however, limit the overall parallel efficiency of the resampling algorithm when, for higher numbers of processes, the execution time for the parallel parts reaches that of the non-parallel parts. To minimize the memory requirements of the algorithm, a block structure for the matrix-matrix product in line 13 can be implemented. In this case, only a small number of rows of Matrix $\mathbf{L_p}$ is allocated and only the corresponding rows of $\mathbf{V_p}$ are updated at a time.

7.3.3 EnKF

The parallel EnKF analysis algorithm for a domain-decomposed ensemble matrix $\mathbf{X}$ is shown as algorithm 7.8. In comparison to the mode-decomposed algorithm, less communication operations are required in the case of domain-decomposition. In particular, there is no need to gather the information on the full ensemble matrix. The operations on the ensemble matrix are completely parallelized.

The information on the full matrix $\mathbf{T1} \in \mathbb{R}^{m \times N}$ is required for the computation of the matrices **T3** and **T6**. Thus, **T1** is initialized on each process using an allgather operation in line 12. Also matrix $\mathbf{D} \in \mathbb{R}^{m \times N}$, which holds the ensemble of residuals, is fully initialized by an allgather operation (line 20). Using the gathered matrices, the computations of **T3** and **T6**, the call to the subroutine *RplusA*, and the solver step to obtain **B** are performed equally by each process. These non-parallelized operations, together with the allgather operations on **T1** and **B** can be expected to limit the overall parallel efficiency of the domain-decomposed EnKF analysis algorithm. Compared with the mode-decomposed variant given as algorithm 7.3, the amount of communicated data is smaller in the domain-decomposed variant. The computations of **B** and **T6**, which are conducted by each process in the case of domain-decomposition are parallelized in the mode-decomposed algorithm. Thus, it is not obvious which of the decomposition variant will yield the better parallel efficiency. Since this depends on the ratio of computation to communication performance, it will depend on the computer architecture on which the algorithms will be executed.

The domain decomposition of the observation space is controlled by the user by, e.g., providing the implementations of the measurement operator. For consistency, the two allgather operations in the domain-decomposed EnKF analysis algorithm are implemented as subroutines to allow the user to modify them. The ordering of matrix rows used for the allgather operation does not need to follow that of the actual domain-decomposition. This fact can simplify the implementation, e.g. in the case of an irregularly decomposed grid in which the sub-states on the processes do not correspond to single blocks in the global state vector. Despite this, the allgather operations in lines 12 and 20 can gather the sub-vectors as single blocks. In this case, consistency is assured by gathering the matrices **T1** and **D** with the same ordering (This is actually assured by performing it by the same subroutine). In addition the subroutine *RplusA* has to be consistent with the gathering order. Ensuring this, the final ensemble update in line 27 will be consistent since the line ordering in matrices $\mathbf{T4_p}$ and **B** is equal.

Subroutine EnKF_Analysis_Domain($step$,n_p,N,$\mathbf{X_p}$)
int $step$ {time step counter,input}
int n_p {state dimension on local domain, input}
int N {ensemble size, input}
real $\mathbf{X_p}(n_p, N)$ {local ensemble matrix, input/output}
real $\mathbf{T1}, \mathbf{T3}, \mathbf{T6}, \mathbf{D}, \mathbf{B}$ {fields to be allocated}
real $\mathbf{T1_p}, \mathbf{t2_p}, \mathbf{t4_p}, \mathbf{T5_p}, \mathbf{D_p}, \mathbf{x_p}$ {fields to be allocated}
int m_p {dimension of local observation vector}
int m {dimension of global observation vector}
int i {ensemble loop counter}

1: call Get_Dim_Obs($step, m_p$) {get observation dimension, user supplied}
2: allreduce summation of m from m_p {global MPI operation}
3: Allocate fields: $\mathbf{T1}(m, N), \mathbf{T3}(m, m), \mathbf{T6}(N, N), \mathbf{D}(m, N), \mathbf{B}(m, N)$,
4: $\mathbf{T1_p}(m_p, N), \mathbf{t2_p}(m_p), \mathbf{t4_p}(m_p), \mathbf{T5_p}(n_p, N), \mathbf{D_p}(m_p, N), \mathbf{x_p}(n_p)$

5: **for** i=1,N **do**
6: call Measurement_Operator($step, n_p, m_p, \mathbf{X_p}(:, i), \mathbf{T1_p}(:, i)$) {local domain}
7: **end for**
8: $\mathbf{t2_p} \leftarrow N^{-1} \sum_{i=1}^{N} \mathbf{T1_p}(:, i)$ {mean of projected ensemble for local domain}
9: **for** i=1,N **do**
10: $\mathbf{T1_p}(:, i) \leftarrow \mathbf{T1_p}(:, i) - \mathbf{t2_p}$ {local domain}
11: **end for**
12: allgather $\mathbf{T1}$ from $\mathbf{T1_p}$ {global MPI operation}
13: $\mathbf{T3} \leftarrow (N-1)^{-1}\ \mathbf{T1}\ \mathbf{T1}^T$ {full matrix-matrix product on each process}
14: call RplusA($step$,m,$\mathbf{T3}$) {by each process}

15: call Enkf_Obs_Ensemble($step$,m_p,N,$\mathbf{D_p}$) {local ensemble of observations}
16: **for** i=1,N **do**
17: call Measurement_Operator($step, n_p, m_p, \mathbf{X_p}(:, i), \mathbf{t4_p}$) {local domain}
18: $\mathbf{D_p}(:, i) \leftarrow \mathbf{D_p}(:, i) - \mathbf{t4_p}$ {ensemble of residuals for local domain}
19: **end for**
20: allgather $\mathbf{D}$ from $\mathbf{D_p}$ {global MPI operation}

21: solve $\mathbf{T3}\ \mathbf{B} = \mathbf{D}$ for $\mathbf{B}$ {Get representer amplitudes on each process}
22: $\mathbf{T6} \leftarrow \mathbf{T1}^T\ \mathbf{B}$ {full matrix-matrix product on each process}
23: $\mathbf{x_p} \leftarrow N^{-1} \sum_{i=1}^{N} \mathbf{X_p}(:, i)$ {ensemble mean state for local domain}
24: **for** i=1,N **do**
25: $\mathbf{T5_p}(:, i) \leftarrow \mathbf{X_p}(:, i) - \mathbf{x_p}$ {local domain}
26: **end for**
27: $\mathbf{X_p} \leftarrow \mathbf{X_p} + (N-1)^{-1}\ \mathbf{T5_p}\ \mathbf{T6}$ {matrix-matrix product of type 2}
28: De-allocate local analysis fields

Algorithm 7.8: Structure of the parallel filter analysis routine for the EnKF algorithm for domain decomposed states. It uses the representer update variant for a non-singular matrix $\mathbf{T5}$. Matrix $\mathbf{T1_p}$ is not allocated but stored in $\mathbf{D_p}$. Analogously the contents of the arrays $\mathbf{B}$ and $\mathbf{t4}$ is stored respectively in $\mathbf{D}$ and $\mathbf{t2}$. Line 27 can be implemented with a block formulation. Then only some rows of $\mathbf{T5_p}$ need to be allocated.

7.3.4 SEIK

As in the case of mode-decomposed ensemble and mode matrices, the analysis algorithm of the SEIK filter for domain-decomposition is very similar to that of the SEEK filter. The parallel SEIK analysis algorithm for domain-decomposition is shown as algorithm 7.9. Again we discuss the differences to the SEEK algorithm.

For domain-decomposition, a process knows the full state ensemble for its local domain. Thus, the computation of ensemble means does not require any MPI operations. Accordingly, the product of matrix $\mathbf{T1_p}$ with matrix $\mathbf{T}$ in line 7 involves no communications of data. The same is true for the computation of the ensemble mean in line 13 and the application of $\mathbf{T}$ to $\mathbf{t5}$ in line 20. Due to this, the amount of communicated data is equal for the analysis algorithms of SEEK and SEIK in the case of domain-decomposition. The algorithm contains several operations which are executed without parallelization. These are the initializations of $\mathbf{G}$ and $\mathbf{Uinv}$, the solver step for $\mathbf{t5}$, and the computation of $\mathbf{t6}$. Most costly will be the solver step for $\mathbf{t5}$ in line 19, since it involves the inversion of $\mathbf{Uinv} \in \mathbb{R}^{r \times r}$. These operations, together with the required communication operations, will limit the parallel efficiency of the domain-decomposed analysis. The parallel efficiency will be, however, better than in the case of mode-decomposition, since there the amount of communicated data is much higher than for domain-decomposition.

For domain-decomposed states, the resampling algorithm of SEIK, shown as algorithm 7.10, has the benefit that no communication operations are required at all. The operations on the small $r \times r$ and $r \times (r+1)$ matrices are performed equally by all processes. They can be expected to require negligible time compared with the computation of the new ensemble states. The operations on the ensemble matrix are fully parallelized. Hence, the domain-decomposed resampling algorithm of SEIK can be expected to show a nearly ideal speedup. To reduce the required memory, we implement the ensemble transformation in line 11 using a block formulation. It is analogous to the block structure described for the SEEK resampling algorithm.

7.3.5 Comparison of Communication and Memory Requirements

Table 7.3 summarizes the size of the communicated arrays in the domain-decomposed filter algorithms. The numbers assume that no communication is performed in the implementation of the measurement operator and in the subroutine *RinvA*.

Since we have usually $n \gg m > N, r$ for realistic large scale models, it is obvious from table 7.3, that with domain decomposition significantly less data has to be communicated between processes. The smallest amount is in the SEIK algorithm. Its analysis algorithm communicates only two arrays of sizes $r \times r$ and r. The resampling algorithm of SEIK is even executed without any communication of data. The largest amount will be in the EnKF algorithm, since here arrays involving the dimension m are communicated.

Comparing the mode-decomposed algorithms (7.1 to 7.5) with the algorithms using domain decomposition (7.6 to 7.10), the smaller memory requirements of the domain-decomposed filter algorithms become visible. Using domain-decomposition, all arrays

Subroutine SEIK_Analysis_Domain($step$,n_p,N,$\mathbf{x_p}$,$\mathbf{Uinv}$,$\mathbf{X_p}$)
int $step$ {time step counter,input}
int n_p {state dimension on local domain, input}
int N {ensemble size, input}
real $\mathbf{x_p}(n_p)$ {local state estimate, output}
real $\mathbf{Uinv}(r,r)$ {inverse eigenvalue matrix, output}
real $\mathbf{X_p}(n_p,N)$ {local ensemble matrix, input/output}
real $\mathbf{t4},\mathbf{t5},\mathbf{t6},\mathbf{G},\mathbf{\Delta Uinv},\mathbf{y_p},\mathbf{d_p}$, {fields to be allocated}
real $\mathbf{T1_p},\mathbf{T2_p},\mathbf{T3_p},\mathbf{t4_p},\mathbf{\Delta Uinv_p}$ {fields to be allocated}
int m_p {dimension of local observation vector}
int i {ensemble loop counter}
int r {rank of covariance matrix, $r = N-1$}

1: call Get_Dim_Obs($step, m_p$) {get observation dimension, user supplied}
2: Allocate fields: $\mathbf{t4}(r), \mathbf{t5}(r), \mathbf{t6}(N), \mathbf{G}(r,r), \mathbf{\Delta Uinv}(r,r), \mathbf{y_p}(m_p), \mathbf{d_p}(m_p)$,
3: $\mathbf{T1_p}(m_p,N), \mathbf{T2_p}(m_p,r), \mathbf{T3_p}(m_p,r), \mathbf{t4_p}(r), \mathbf{\Delta Uinv_p}(r,r)$

4: **for** i=1,N **do**
5: call Measurement_Operator($step, n_p, m_p, \mathbf{X_p}(:,i), \mathbf{T1_p}(:,i)$) {local domain}
6: **end for**
7: $\mathbf{T2_p} \leftarrow \mathbf{T1_p}\ \mathbf{T}$ {implemented with $\mathbf{T}$ as operator}
8: call RinvA($step, m, r, \mathbf{T2_p}, \mathbf{T3_p}$) {operate only on local domain}
9: $\mathbf{G} \leftarrow (N^{-1}(\mathbf{T}^T\ \mathbf{T})^{-1})$ {by each process; implemented as direct initialization}
10: $\mathbf{\Delta Uinv_p} \leftarrow \mathbf{T2_p}^T\mathbf{T3_p}$ {matrix-matrix product of type 3}
11: allreduce summation of $\mathbf{\Delta Uinv}$ from $\mathbf{\Delta Uinv_p}$ {global MPI operation}
12: $\mathbf{Uinv} \leftarrow \mathbf{G} + \mathbf{\Delta Uinv}$ {by each process}

13: $\mathbf{x_p} \leftarrow N^{-1}\sum_{i=1}^{N}\mathbf{X_p}(:,i)$ {get ensemble mean state for local domain}
14: call Measurement_Operator($step, n_p, m_p, \mathbf{x_p}, \mathbf{d_p}$) {user supplied}
15: call Measurement($step, m_p, \mathbf{y_p}$) {user supplied}
16: $\mathbf{d_p} \leftarrow \mathbf{y_p} - \mathbf{d_p}$

17: $\mathbf{t4_p} \leftarrow \mathbf{T3_p}^T\mathbf{d_p}$ {matrix-matrix product of type 3}
18: allreduce summation of $\mathbf{t4}$ from $\mathbf{t4_p}$ {global MPI operation}
19: solve $\mathbf{Uinv}\ \mathbf{t5} = \mathbf{t4}$ for $\mathbf{t5}$ {by each process}
20: $\mathbf{t6} \leftarrow \mathbf{T}\ \mathbf{t5}$ {implemented with $\mathbf{T}$ as operator}
21: $\mathbf{x_p} \leftarrow \mathbf{x_p} + \mathbf{X_p}\ \mathbf{t6}$ {matrix-vector product of type 2}
22: De-allocate local analysis fields

Algorithm 7.9: Structure of the parallel filter analysis routine for the SEIK algorithm for domain decomposed states. The arrays $\mathbf{T2_p}$ and $\mathbf{G}$ are not allocated but stored respectively in $\mathbf{T1_p}$ and $\mathbf{Uinv}$. Analogously, the contents of $\mathbf{t5}$ are stored in $\mathbf{t4}$.

Subroutine SEIK_Resample_Domain(n_p,N,$\mathbf{x_p}$,$\mathbf{Uinv}$,$\mathbf{X_p}$)
int n_p {state dimension on local domain, input}
int N {ensemble size, input}
real $\mathbf{x_p}(n_p)$ {state analysis vector, input}
real $\mathbf{Uinv}(r,r)$ {inverse eigenvalue matrix, input}
real $\mathbf{X_p}(n_p,N)$ {ensemble matrix, input/output}
real $\mathbf{T1},\mathbf{T2},\mathbf{\Omega}^T,\mathbf{C},\mathbf{T3_p}$ {fields to be allocated}
int r {rank of covariance matrix, $r=N-1$}

1: Allocate fields: $\mathbf{T1}(r,N),\mathbf{T2}(N,N),\mathbf{\Omega}^T(r,N),\mathbf{C}(r,r),\mathbf{T3_p}(n_p,N)$

2: Cholesky decomposition: $\mathbf{Uinv}=\mathbf{C}\,\mathbf{C}^T$ {by each process}
3: initialize $\mathbf{\Omega}^T$ {by each process}
4: solve $\mathbf{C}^T\mathbf{T1}=\mathbf{\Omega}^T$ for $\mathbf{T1}$ {by each process}
5: $\mathbf{T2}\leftarrow\mathbf{T}\,\mathbf{T1}$ {implemented with $\mathbf{T}$ as operator}
6: **for** i=1,N **do**
7: $\mathbf{T3_p}(:,i)\leftarrow\mathbf{X_p}(:,i)$
8: $\mathbf{X_p}(:,i)\leftarrow\mathbf{x_p}$
9: **end for**
10: $\mathbf{X_p}\leftarrow\mathbf{X_p}+N^{1/2}\,\mathbf{T3_p}\,\mathbf{T2}$ {matrix-matrix product of type 2}
11: De-allocate local analysis fields

Algorithm 7.10: Structure of the parallel resampling routine for the SEIK algorithm for domain decomposed states. The matrix $\mathbf{T1}$ is never allocated in the program. Its contents are stored in $\mathbf{\Omega}^T$. Lines 6 to 10 can be implemented with a block formulation. Then only some rows of $\mathbf{T3_p}$ are allocated.

Table 7.3: Sizes of arrays involved in global MPI operations in the analysis and resampling phases of the SEEK and SEEK algorithms and in the analysis phase of the EnKF algorithm for domain-decomposed states. Next to the matrix size, the name of the matrix is given as well as the information whether the MPI operation is an allgather (g) or allreduce (r) operation.

	EnKF	SEEK	SEIK
analysis	mN ($\mathbf{T1}$, g) mN ($\mathbf{D}$, g) 1 (m, r)	r^2 ($\mathbf{\Delta Uinv}$, r) r ($\mathbf{t3}$, r)	r^2 ($\mathbf{\Delta Uinv}$, r) r ($\mathbf{t4}$, r)
resampling		r^2 ($\mathbf{T1}$, r)	

involving the state dimension n are distributed for all three filters. In SEEK and SEIK also all arrays involving the dimension m are distributed. In contrast to this, there are only small memory overheads. They are caused by arrays involving the ensemble size N which have to be added in comparison to the serial algorithms discussed in section 3.3. Since the ensemble size is typically much smaller than the dimensions n and m, the domain-decomposed SEEK and SEIK algorithms are scalable in terms of memory requirements. In the EnKF, the situation is more problematic. The arrays $\mathbf{T1} \in \mathbb{R}^{m\times N}$, $\mathbf{D} \in \mathbb{R}^{m\times N}$, and $\mathbf{T3} \in \mathbb{R}^{m\times m}$ are fully allocated on each process. Also one array of size $m_p \times N$ ($\mathbf{D_p}$) has to be added in comparison to the serial algorithm. If large observational data sets have to be assimilated, matrix **T3** will dominate the memory requirements.

7.4 Localized Filter Analyses

The parallelization schemes presented above are solely based on a reformulation of the serial algorithms to distribute fields and work over the available processes. Thus, no approximations are involved. Slightly different results in the analysis might occur due to numerical reasons caused by a different order in parallelized summations compared with a sum computed by a single process. The analyses algorithms of the filters are spatially global, since long range covariances might exist. In addition, the analysis and resampling phases are global over the state or mode ensembles, since weighted averages of the ensemble members are computed. Due to this, several global MPI operations are performed in the analysis and resampling phases of the filter algorithms. These global communication operations will always limit the parallel efficiency of the filter algorithms.

When we consider the filter algorithms developed for domain decomposed states, the amount of communicated data is smaller than their mode-decomposed counterparts. The amount of data communicated in the SEEK and SEIK filters is much lower than in the EnKF. The analysis and resampling algorithms of SEEK and SEIK are formulated such that all operations on the state space and the observation space are decomposed. These algorithms are global only in the error space of dimension r. Hence, with domain-decomposed states, communication operations are required only for fields in the error space. Since all operations in the state space and the observation space are parallelized without communication of data, a further localization of the SEEK and SEIK algorithms does not appear to be necessary.

The situation is different for the EnKF with domain-decomposition. The EnKF computes the weights for the ensemble update in the observation space of dimension m. In particular, the computation of **T3** in line 13 of algorithm 7.8 and the solver step for the representer amplitudes **B** in line 25 are costly. These operations are especially problematic since they are not parallelized and therefore executed by each process. Thus, they reduce the parallel efficiency of the algorithm. The efficiency is further diminished by the allgather operations in lines 12 and 20.

To reduce the dimension of the observation vector in the analysis algorithm, it is possible to formulate a localized analysis algorithm. This is based on the assumption

that observations have negligible influence for the analysis update of a certain grid point if they have a large distance to this grid point. In this case, only observations within a certain distance from the grid point need to be taken into account for the analysis of the state of this location. The local analysis is an approximation to the global analysis, but it is motivated by the fact that long range covariances in the matrix $\tilde{\mathbf{P}}$, which is represented by the ensemble, are very noisy and their information contents will be negligible. This topic has been discussed, e.g., by Houtekamer and Mitchell [34]. To perform the localization, Houtekamer and Mitchell [36] filtered the covariance matrix $\tilde{\mathbf{P}}$ by a Schur product, i.e. an element-wise product, with a matrix representing correlations of local support. This technique has also been used by Keppenne and Rienecker [45] who apply the localization for data assimilation in an parallelized ocean general circulation model.

The effect of the introduced smoothing and down-weighting of observations at intermediate distances and neglecting of remote observations has been examined by Hamill et al. [30]. Their results showed that for small ensembles the cut-off radius for the observations should be rather small to obtain a minimal estimation error. Typically an optimal radius which minimizes the estimation error can be determined. On the other hand Mitchell and Houtekamer [56] showed that the localization causes imbalance in the analysis state of a primitive equation model. This imbalance increases with decreasing cut-off radius. Evensen [18] also argued against a filtering of the covariances, since this will introduce spurious and nondynamical modes in the analysis. Evensen, on the other hand argues in favor of a local analysis since this increases the degrees of freedom in the update of the ensemble states. I.e. each local domain will be updated using a different combination of the ensemble states. This will eventually lead to a state estimate with smaller estimation errors than a global analysis update.

We will derive equations for the local analysis which do not use a Schur product to filter and localize the covariances. Our formulation just neglects observations beyond the cut-off radius. For the filtering by a Schur product this would correspond to a step function of the correlations. In this respect, our formulation follows that suggested by Evensen [18]. Figure 7.3 visualizes the domain decomposition for a localized analysis in a structured rectangular grid. We intent to update the sub-domain $\mathcal{S}$. When we assume direction dependent cut-off radii $(r_1,\ r_2)$, the influence region of observations for the upper right edge of $\mathcal{S}$ is given by the ellipse $\mathcal{C}$. The region $\mathcal{D}$ shaded in light grey is the observation influence region for the whole sub-domain $\mathcal{S}$. In finite difference models with structured grids, for simplicity the rectangular region $\tilde{\mathcal{D}}$ could be chosen as influence region. This localization differs from that suggested by Ott et al. [58]. While Ott et al. use coinciding domains for the sub-domain $\mathcal{S}$ in which the state is updated and the observation domain $\mathcal{D}$ we assume that $\mathcal{D}$ contains all observations within a certain distance from the grid points in $\mathcal{S}$.

To obtain a mathematical formulation for the localization, we consider the basic analysis equations 2.41 and 2.42 of the EnKF algorithm. Omitting the time index k, the global analysis equations for each ensemble state $\{\mathbf{x}^{(\alpha)},\ \alpha = 1, \ldots, N\}$ are:

$$\mathbf{x}^{a(\alpha)} = \mathbf{x}^{f(\alpha)} + \tilde{\mathbf{K}}\left(\mathbf{y}^{o(\alpha)} - \mathbf{H}\mathbf{x}^{f(\alpha)}\right) \tag{7.1}$$

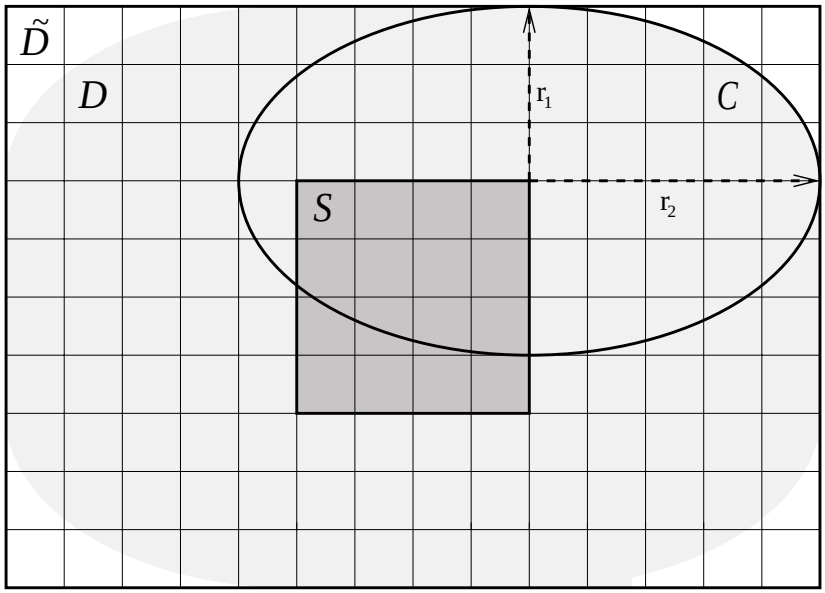


Figure 7.3: Domain decomposition for a localized analysis in a structured rectangular grid (Following the representation by Keppenne and Rienecker [45]). Region $\mathcal{S}$ is the sub-domain in which the state is updated. The ellipse $\mathcal{C}$ marks the influence region of observations for the grid point at the upper right edge of region $\mathcal{S}$. $\mathcal{C}$ is defined by the cut-off radii r_1 and r_2. The region $\mathcal{D}$ shaded in light grey marks the influence region of the observations for the whole region $\mathcal{S}$.

with

$$\tilde{\mathbf{K}} = \tilde{\mathbf{P}}^f \mathbf{H}^T \Big(\mathbf{H} \tilde{\mathbf{P}}^f \mathbf{H}^T + \mathbf{R} \Big)^{-1} . \tag{7.2}$$

Now let $\mathbf{S}_\sigma$ be a linear operator which reduces a global state vector $\mathbf{x}$ of dimension n to its local part $\mathbf{x}_\sigma$ of dimension $n_\sigma < n$ in the sub-domain $\mathcal{S}_\sigma$. The subscript σ denotes the set of parameters which specify the sub-domain. For simplicity, we assume here that the sub-domain is specified by the spatial position $\mathbf{l}$ of its center as well as its extent $\mathbf{r}_\sigma$ in the spatial directions. Then we can write the analysis of the local state as

$$\mathbf{x}_\sigma^{a(\alpha)} := \mathbf{S}_\sigma \mathbf{x}^{a(\alpha)} = \mathbf{S}_\sigma \mathbf{x}^{f(\alpha)} + \mathbf{S}_\sigma \tilde{\mathbf{K}} \Big(\mathbf{y}^{o(\alpha)} - \mathbf{H} \mathbf{x}^{f(\alpha)} \Big) . \tag{7.3}$$

Let $\mathbf{D}_\delta$ be a linear operator which reduces a global observation vector $\mathbf{y}$ of dimension m to its local part $\mathbf{y}_\delta$ in the sub-domain $\mathcal{D}_\delta$. The subscript δ denotes the set of parameters which specify the sub-domain in the global observation domain. We assume that $\mathcal{D}_\delta$ is centered at the same spatial location $\mathbf{l}$ as the state sub-domain $\mathcal{S}_\sigma$ but the extent of $\mathcal{D}_\delta$ will be different from that of $\mathcal{S}_\sigma$. Now we can write the analysis for the local state using only observations from domain $\mathcal{D}_\delta$ as

$$\mathbf{S}_\sigma \mathbf{x}^{a(\alpha)} = \mathbf{S}_\sigma \mathbf{x}^{f(\alpha)} + \tilde{\mathbf{K}}_{\sigma\delta} \Big(\mathbf{D}_\delta \mathbf{y}^{o(\alpha)} - \mathbf{D}_\delta \mathbf{H} \mathbf{x}^{f(\alpha)} \Big) \tag{7.4}$$

with

$$\tilde{\mathbf{K}}_{\sigma\delta} \;=\; \mathbf{S}_\sigma \tilde{\mathbf{P}}^f \mathbf{H}^T {\mathbf{D}_\delta}^T \Big(\mathbf{D}_\delta \mathbf{H} \tilde{\mathbf{P}}^f \mathbf{H}^T \mathbf{D}_\delta^T + \mathbf{D}_\delta \mathbf{R} \mathbf{D}_\delta^T \Big)^{-1} . \tag{7.5}$$

The application of the operator $\mathbf{D}_\delta$ amounts to the neglect of observations which are beyond the sub-domain $\mathcal{D}_\delta$.

Now we define the measurement operator $\mathbf{H}_\delta := \mathbf{D}_\delta \mathbf{H}$ which projects a (global) state vector onto the local observation domain $\mathcal{D}_\delta$. In addition, we define the observation error covariance matrix in $\mathcal{D}_\delta$ as $\mathbf{R}_\delta := \mathbf{D}_\delta \mathbf{R} \mathbf{D}_\delta^T$. With these definitions the local analysis equations for the EnKF are

$$\mathbf{x}_\sigma{}^{a(\alpha)} = \mathbf{x}_\sigma{}^{f(\alpha)} + \tilde{\mathbf{K}}_{\sigma\delta} \left(\mathbf{y}_\delta{}^{o(\alpha)} - \mathbf{H}_\delta \mathbf{x}^{f(\alpha)} \right) \tag{7.6}$$

with

$$\tilde{\mathbf{K}}_{\sigma\delta} = \mathbf{S}_\sigma \tilde{\mathbf{P}}^f \mathbf{H}_\delta{}^T \left(\mathbf{H}_\delta \tilde{\mathbf{P}}^f \mathbf{H}_\delta^T + \mathbf{R}_\delta \right)^{-1} . \tag{7.7}$$

For the local analysis these equations replace equations (2.41) and (2.42) of the global analysis. The local representer formulation follows as the local alternative to equations (2.46) and (2.47) as

$$\mathbf{x}_\sigma^{a(\alpha)} = \mathbf{x}_\sigma^{f(\alpha)} + \mathbf{S}_\sigma \tilde{\mathbf{P}}^f \mathbf{H}_\delta^T \mathbf{b}^{(\alpha)\delta} \tag{7.8}$$

and

$$(\mathbf{H}_\delta \tilde{\mathbf{P}}^f \mathbf{H}_\delta{}^T + \mathbf{R}_\delta) \mathbf{b}_\delta^{(\alpha)} = \mathbf{y}_\delta^{o(\alpha)} - \mathbf{H}_\delta \mathbf{x}^{f(\alpha)} . \tag{7.9}$$

Based on equations (7.6) and (7.7, we can also reformulate) the ensemble computation of the matrices $\tilde{\mathbf{P}}^f \mathbf{H}^T$ and $\mathbf{H} \tilde{\mathbf{P}}^f \mathbf{H}^T$ (equations (2.48) and (2.49)) for the local analysis. These are:

$$\mathbf{S}_\sigma \tilde{\mathbf{P}}^f \mathbf{H}_\delta^T = \frac{1}{N-1} \sum_{\alpha=1}^{N} ({}^{(\alpha)}\mathbf{x}_\delta^f - \overline{\mathbf{x}_\delta^f})[\mathbf{H}_\delta({}^{(\alpha)}\mathbf{x}^f - \overline{\mathbf{x}^f})]^T , \tag{7.10}$$

$$\mathbf{H}_\delta \tilde{\mathbf{P}}^f \mathbf{H}_\delta^T = \frac{1}{N-1} \sum_{\alpha=1}^{N} \mathbf{H}_\delta({}^{(\alpha)}\mathbf{x}^f - \overline{\mathbf{x}^f})[\mathbf{H}_\delta({}^{(\alpha)}\mathbf{x}^f - \overline{\mathbf{x}^f})]^T \tag{7.11}$$

These equations can be implemented using the same optimization strategy as for the other parallelized EnKF analysis algorithms. The Algorithm 7.11 shows the algorithm in pseudo code. Apart from the distinction of private and global variables, it is identical to the structure of the serial program shown in algorithm 3.5. In particular, no communications are performed in the analysis routine itself. However, the called subroutines are different from their serial variants. *Get_Dim_Obs* now provides the dimension of the local observation vector $\mathbf{y}_\delta^o$ and *EnKF_Obs_Ensemble* initializes the local observation ensemble $\mathbf{Y}_\delta^o$. Also, *RplusA* adds the local observation error covariance matrix $\mathbf{R}_\delta$. Analogously, the routine *Measurement_Operator* provides a state vector projected on the local observation space on the basis of the global state vector. This routine has as input only a state vector for the local domain. Thus the routine *Measurement_Operator* will involve communications of data from other state sub-domains if the domains $\mathcal{S}_\sigma$ and $\mathcal{D}_\delta$ do not coincide. As long as the local observation domain is smaller than the global observation domain, these communication operations will not involve all processes. The implementation of the localized analysis algorithm

Subroutine EnKF_Analysis_Local($step$,n_p,N,$\mathbf{X_p}$)
int $step$ {time step counter,input}
int n_p {state dimension on local domain, input}
int N {ensemble size, input}
real $\mathbf{X_p}(n_p, N)$ {local ensemble matrix, input/output}
real $\mathbf{T1_p}, \mathbf{t2_p}, \mathbf{T3_p}, \mathbf{t4_p}, \mathbf{T5_p}, \mathbf{T6_p}, \mathbf{D_p}, \mathbf{x_p}$ {fields to be allocated}
int m_p {dimension of observation vector in the local domain}
int i {ensemble loop counter}

1: call Get_Dim_Obs($step, m_p$) {dimension for local observation domain $\mathcal{D}_\delta$}
2: Allocate fields: $\mathbf{T1_p}(m_p, N), \mathbf{t2_p}(m_p), \mathbf{T3_p}(m_p, m_p), \mathbf{t4_p}(m_p)$,
3: $\mathbf{T5_p}(n_p, N), \mathbf{T6_p}(N, N), \mathbf{B_p}(m_p, N), \mathbf{D_p}(m_p, N), \mathbf{x_p}(n_p)$

4: **for** i=1,N **do**
5: call Measurement_Operator($step, n_p, m_p, \mathbf{X_p}(:, i), \mathbf{T1_p}(:, i)$) {in domain $\mathcal{D}_\delta$}
6: **end for**
7: $\mathbf{t2_p} \leftarrow N^{-1} \sum_{i=1}^{N} \mathbf{T1_p}(:, i)$ {in domain $\mathcal{D}_\delta$}
8: **for** i=1,N **do**
9: $\mathbf{T1_p}(:, i) \leftarrow \mathbf{T1_p}(:, i) - \mathbf{t2_p}$ {in observation domain $\mathcal{D}_\delta$}
10: **end for**
11: $\mathbf{T3_p} \leftarrow (N-1)^{-1}\ \mathbf{T1_p}\ \mathbf{T1_p^T}$ {full matrix-matrix product in $\mathcal{D}_\delta$}
12: call RplusA($step$,m_p,$\mathbf{T3_p}$) {in domain $\mathcal{D}_\delta$}

13: call Enkf_Obs_Ensemble($step$,m_p,N,$\mathbf{D_p}$) {ensemble of observations in $\mathcal{D}_\delta$}
14: **for** i=1,N **do**
15: call Measurement_Operator($step, n_p, m_p, \mathbf{X_p}(:, i), \mathbf{t4_p}$) {in domain $\mathcal{D}_\delta$}
16: $\mathbf{D_p}(:, i) \leftarrow \mathbf{D_p}(:, i) - \mathbf{t4_p}$ {ensemble of residuals for domain $\mathcal{D}_\delta$}
17: **end for**

18: solve $\mathbf{T3_p}\ \mathbf{B_p} = \mathbf{D_p}$ for $\mathbf{B_p}$ {in domain $\mathcal{D}_\delta$}
19: $\mathbf{x_p} \leftarrow N^{-1} \sum_{i=1}^{N} \mathbf{X_p}(:, i)$ {ensemble mean state for local domain $\mathcal{S}_\sigma$}
20: **for** i=1,N **do**
21: $\mathbf{T5_p}(:, i) \leftarrow \mathbf{X_p}(:, i) - \mathbf{x_p}$ {in domain $\mathcal{S}_\sigma$}
22: **end for**
23: $\mathbf{T6_p} \leftarrow \mathbf{T1_p^T}\ \mathbf{B_p}$ {in domain $\mathcal{D}_\delta$}
24: $\mathbf{X_p} \leftarrow \mathbf{X_p} + (N-1)^{-1}\ \mathbf{T5_p}\ \mathbf{T6_p}$ {full matrix-matrix product in $\mathcal{S}_\sigma$}
25: De-allocate local analysis fields

Algorithm 7.11: Structure of the local filter analysis routine for the EnKF algorithm using domain decomposed states. This routine applies the representer update variant for a non-singular matrix $\mathbf{T5_p}$. Matrix $\mathbf{T1_p}$ is not allocated but stored in $\mathbf{D_p}$. Analogously, the contents of $\mathbf{B_p}$ and $\mathbf{t4_p}$ is stored respectively in $\mathbf{D_p}$ and $\mathbf{t2_p}$. To avoid the allocation of the full array $\mathbf{T5_p}$, line 24 can be implemented in block formulation.

is independent of the model grid. Thus, it can be also applied for unstructured grids like those which can appear with finite element models.

The local formulation has the benefit that no arrays involving the full observation dimension m need to be allocated. The Matrices $\mathbf{T1_p}$ and $\mathbf{D_p}$ are now of size $m_p \times N$ and matrix $\mathbf{T3_p}$ has only dimension $m_p \times m_p$. The amount of computations is as well reduced in comparison to the domain-decomposed global analysis algorithm 7.8. The matrix-matrix products to compute $\mathbf{T3_p}$ (line 11) and $\mathbf{T6_p}$ (line 23) involve now the dimension m_p instead m. Also, the solver step to obtain the representer amplitudes $\mathbf{B_p}$ (line 18) is computed in the domain $\mathcal{D}_\delta$.

As long as the domains $\mathcal{S}_\sigma$ and $\mathcal{D}_\delta$ do not coincide, the local analysis formulation still requires communication of data. These communication operations are, however, not global and involve less amount of data than the global domain-decomposed formulation of the algorithm. In addition, the localization permits to distribute all computations on observation-related matrices including, e.g., the solver step for the representer amplitudes. Thus, the local algorithm can be expected to show a much better scalability and parallel efficiency than the global algorithm.

7.5 Summary

In this chapter, we examined strategies to parallelize the analysis and resampling phases of the SEEK, EnKF, and SEIK filter algorithms. There are two different parallelization strategies:

1. *Mode-decomposition* – The filter can be parallelized over the modes of the ensemble matrix $\mathbf{X}$ or the mode matrix $\mathbf{V}$. In this case, the matrix is decomposed such that each process holds several columns of $\mathbf{X}$ or $\mathbf{V}$. Since each column of the matrix represents a full model state vector, the filter operates on sub-ensembles of model states. This parallelization strategy of the filter is independent from a possible parallelization of the numerical model used to compute the forecast. Since each ensemble state can be evolved independently from the other states, this parallelization exploits the inherent parallelism of the ensemble forecast.

2. *Domain-decomposition* – The filter can be parallelized by a decomposition of the model domain. In this case each process holds several rows of the matrices $\mathbf{X}$ or $\mathbf{V}$. Thus, each process operates on a full ensemble of model sub-states for the domain owned by this process. With this parallelization strategy, the filter typically applies the same domain-decomposition as the numerical model. Different decompositions for model and filter are possible, but will yield an overhead when the state information is transferred between filter and model. This is due to the reordering of the state information.

We also discussed the implementation of a localized filter analysis for the situation of domain-decomposed states. This localization neglects observations beyond some distance from a model sub-domain. Thus, it reduces the effective dimension of the observation vector. It became evident that a localization is only useful for the EnKF. The

SEEK and SEIK filters operate globally only on the error subspace which is spanned by the ensemble states. Since the error subspace is typically of much lower dimension than the local model domain, the global operations will not significantly limit the parallel efficiency of the algorithms. For the EnKF, the localization reduces the amount of communicated data. In addition, the computations are distributed more evenly among the processes than in the global formulation of the analysis. Thus, the localization will provide a better scalability of the EnKF algorithm compared with a global analysis. We obtained a particularly simple formulation for the implementation of the EnKF analysis routine. The analysis routine is formulated like the serial algorithm discussed in section 3.3 while the localization is entirely handled in the observation-dependent routines which are provided by the user of the algorithm.

For the global algorithms, tables 7.2 and 7.3 showed that significantly less data is communicated if the variant with domain-decomposed states is used. The least amount of communication is necessary for the SEIK filter. In addition, the memory requirements are smaller for the variant with domain-decomposition than with decomposition over the modes of the ensemble matrix. Using domain-decomposed states, all matrices involving the state dimension n or the dimension m of the observation vector are decomposed in the SEEK and SEIK algorithms. This provides scalability of the memory requirements. In the EnKF, all matrices involving the state dimension n are decomposed, too. It is, however, still required to allocate matrices involving the observation dimension m. Thus, the EnKF requires more memory than the SEEK and SEIK algorithms. In addition, the memory requirements do not scale with the number of processes. Scalability of the memory requirements is assured if the localized analysis algorithm is used. In this case, all matrices involving the observation dimension are decomposed and refer only to the local observation domain.

Since the state or ensemble updates of the filter analysis and resampling phases correspond to the computation of weighted averages of the ensemble members, it is much more efficient to store whole ensembles of sub-states on each process than to store sub-ensemble of whole states. Thus, from the algorithmic point of view, the domain-decomposed filter algorithms are superior to the mode-decomposed filters. Most efficient is the domain-decomposed SEIK filter. It decomposes all matrices involving the larger dimensions n and m. Communication operations are only necessary on matrices involving the dimension r of the error subspace. The localized EnKF algorithm will also be efficient. However, this algorithm approximates the analysis by neglecting observations beyond a certain distance.

The different parallel efficiencies of the algorithms, however, will be less important in data assimilation applications if the forecast phase dominates the computation time. In this case, it is important that the ensemble forecast exhibits good parallel efficiency. This issue is discussed in the next chapter in conjunction with the development of a parallel filtering framework.

Chapter 8

A Framework for Parallel Filtering

8.1 Introduction

As we have discussed above, the forecast phase of the EnKF and SEIK filters consists of an evolution of N independent model states. In addition, the evolutions of the modes in the SEEK filter are independent, if a gradient approximation for the linearized model is used. To utilize this natural parallelism of the forecast phase and the parallelization possibilities of the analysis and resampling phases discussed in chapter 7, we develop a framework for parallel data assimilation based on filter algorithms. The framework defines an application program interface (API) which permits to combine a filter algorithm with a numerical model. The filter algorithm is attached to the model with minimal changes of the model source code itself. The API permits to switch easily between different filter algorithms. Parts of the data assimilation program which are specific to the model or refer to observations are hold in separate subroutines. These have to be provided by the user of the framework such that they can be called in the filter routines via the API. Accordingly, no changes to the filter routines themselves are required when a data assimilation system is implemented utilizing the filter framework. Thus, it is possible to compile the filter routines separately from the data assimilation program and to distribute them as a program library.

Existing interface structures are the programs SESAM [75] and PALM [60]. SESAM is based on UNIX shell scripts which control the execution of separated program executables. This structure requires that all data transfers between different programs in the data assimilation system are performed using disk files. SESAM has the benefit that no changes to the model source code are required, since the structure of the data assimilation system is defined externally to the model. The problem of data exchanges between the model and the filter program, i.e. the analysis and resampling phases, is shifted to the problem of a consistent format of the data files. Eventually the disk read/write routines have to be changed in the model or file transformation programs are required. The system does not allow for parallel model tasks, as it is based on shell scripts. Furthermore, the overall performance in terms of computation time will not be optimal, since disk operations are extremely slow in comparison to memory operations.

The concept of the PALM system is quite different. This coupler is based on an abstract flow chart representation of data assimilation systems [48]. PALM provides a graphical user interface (GUI) in which the data assimilation system is assembled from separated subroutines following the flow chart representation. In addition, PALM provides a library of algebraic routines. These are prepared for the PALM system and can be used directly in the GUI. Subsequently, an executable program is compiled within the PALM framework according to a structure file written by the GUI. The structure of PALM is highly flexible. It requires, however, that subroutines are prepared to be used with PALM. For this, the routines are extended by a definition header. In addition, subroutine calls for data transfers are added. In PALM, the construction of the whole program including the data assimilation algorithm is shifted to the GUI.

The data assimilation framework which we present in this chapter is less abstract and flexible than PALM. On the other hand, the chosen structure gives more control to the user who attaches the filters to the model source code. The calls to the filter interface routines are added directly to the source code of the model. The filter algorithms are fully implemented and optimized using library routines for algebraic operations. We use the BLAS and LAPACK libraries which are provided by the computer vendor, since these are typically highly optimized for the used computer system. There is no need to modify the filter algorithms or to assemble single routines to obtain a working data assimilation algorithm. In addition, the execution of the program is controlled from within the model source code, which is extended to perform data assimilation. The control is not shifted to an exterior environment as in PALM. In discussions with oceanographers, these future users apparently prefer a structure in which the physical model remains the essential part of the data assimilation program and the filter is attached to the model. A structure which passes the model to a coupler interface which controls the program execution appeared to be accepted less. Such a structure was also used for the implementation which we presented in section 3.3. There the control was given to the filter routines after initializing the model. The time stepper of the model itself was called as a subroutine.

There are two different process configurations for the framework. The filter routines can be either executed by (some of) the model processes or disjoint process sets for the filter and model routines can be used. Thus, after introducing the general structure of the framework in section 8.2, we discuss separately the framework structures for two different process configurations in sections 8.3 and 8.4. In both cases, we introduce the API. Further, we discuss possible configurations of the required MPI communicators and explain the execution structures of the framework. Subsequently, we consider in section 8.5 the issue to define the transition between the state vector notation of the filter routines and the physical fields of the model.

8.2 General Considerations

For the development of the framework, we base on the following considerations:

- The numerical model is independent from the routines of the filter algorithms. The model source code should be changed as little as possible when combining the filters with the model.

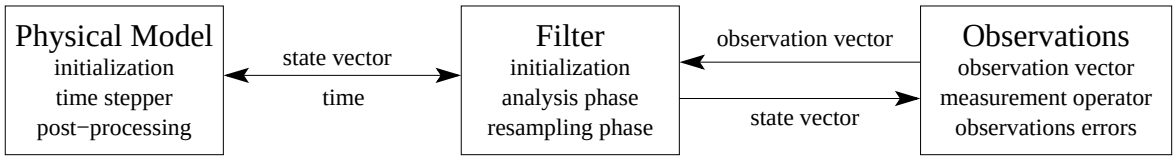


Figure 8.1: Logical parts of the data assimilation problem.

- The filter source code is independent from the model. It solely operates on model state vectors, not on the physical fields of the model.

- The observations are independent both from the numerical model and from the filter. The filter routines require information on the observations (observation vector, measurement operator, observation error covariance matrix) in the analysis phase. The model does not need information about the observations. To implement the measurement operator, however, information on the structure of the state vector is necessary. The physical meaning of the entries (velocities, temperatures, etc.) and their spatial location in the model mesh has to be known. Since the routines which initialize the state ensembles also require this information, it can be shared between the ensemble initialization routines and the implementation of the measurement operator using Fortran modules. The framework can be logically partitioned into three parts as is sketched in figure 8.1. The transfer of information between the model and the filter as well as between the filter and the observations is performed via the API.

- The framework has to allow for the execution of multiple concurrent model evolutions, each of these can be parallelized itself and thus be executed by multiple processes. Both, the parallelization of the model itself and the number of parallel model tasks have to be specified by the user.

- Like the model, the filter routines can be executed in parallel, too. We have discussed the parallelization of the filter routines in chapter 7.

- The filter routines can either by executed by (some of) the processes used for the model evolutions or by a set of processes which is disjoint from the set of model processes.

To combine a filter algorithm with a numerical model in order to obtain a data assimilation program, we consider the 'typical' structure of a model which computes the time evolution of several physical fields. These can be, for example, the temperature and salinity fields in a modeled ocean. The 'typical' structure is depicted in figure 8.2a. In the initialization phase of the program, the mesh for the computations is generated. Also the physical fields are initialized. Subsequently, the evolution is performed. Here *nsteps* time steps of the model fields are computed. These take into account boundary conditions as well as external forcing fields, like e.g. wind fields over

the ocean. At certain time-step intervals, some fields are typically written into disk files and diagnostic quantities are computed. Having completed the evolution some post-processing operations can be performed.

The structure of the data assimilation program with attached filter is shown in figure 8.2b. To initialize the filter framework, a routine *Filter_Init* is added to the initialization part of the program. Here the arrays required for the filter, like the ensemble matrix $\mathbf{X}$, the mode matrix $\mathbf{V}$ or matrix $\mathbf{U}$ of the SEEK filter are allocated. Subsequently, the state estimate $\mathbf{x}_0^a$ and the state ensemble or mode matrices are initialized. The major logical change when combining a filter algorithm with the model source code is that a sequence of independent evolutions has to be computed. This can be achieved by enclosing the time stepping loop by an unconditioned outer loop which is controlled by the filter algorithm. For each evolution the model obtains a model state from the filter algorithm together with the number of time steps to be performed. To enable the consistent application of time dependent forcing in the model the filter also provides the model time at the beginning of the evolution phase. The user has to assure that the evolutions are really independent. Thus, any re-used fields must be newly initialized. In the framework, the model state, the model time (t), and the number of time steps ($nsteps$) are provided by calling a subroutine *Get_State* before the time stepping loop is entered. A value of $nsteps = 0$ uniquely determines that no stepping has to be performed. Thus, this setting is used as an exit-condition within the unconditioned outer loop. After the time stepping loop a subroutine *Put_State* is inserted into the model source code. In this routine the evolved model fields are stored back as a column of the ensemble state matrix of the filter. If the ensemble forecast has not yet finished, no further operations are performed in the routine *Put_State*. When all model states of the current forecast phase are evolved, *Put_State* executes the analysis and resampling phases of the chosen filter algorithm.

For the parallelized version of the data assimilation program, a further change to the model source code concerning the configuration of MPI communicators is required. For MPI-parallelized models there is typically a single model task which operates in the global MPI communicator *MPI_COMM_WORLD*. To allow for multiple model tasks which are executed concurrently, the global communicator has to be replaced by a communicator of disjoint process sets in which each of the model tasks operates. Thus, a communicator *COMM_MODEL* consisting of N_m disjoint process sets has to be generated. In the model source code, the reference to *MPI_COMM_WORLD* has to be replaced by *COMM_MODEL*. Next to the communicator for the model a communicator *COMM_FILTER* has to be created defining the processes which execute the filter routines. To couple the filter processes with the model tasks another communicator *COMM_COUPLE* is required. Using this communicator, data is transfered between the filter and model parts of the data assimilation framework.

The configuration of the MPI communicators is dependent on the choice whether the filter routines are executed by some of the model processes or on a set of processes which is disjoint from the set of model processes. In addition, the API for calling the subroutines *Filter_Init*, *Get_State*, and *Put_State* depends on this choice of the process configuration. For this reason, we discuss the two different configurations separately in the following sections.

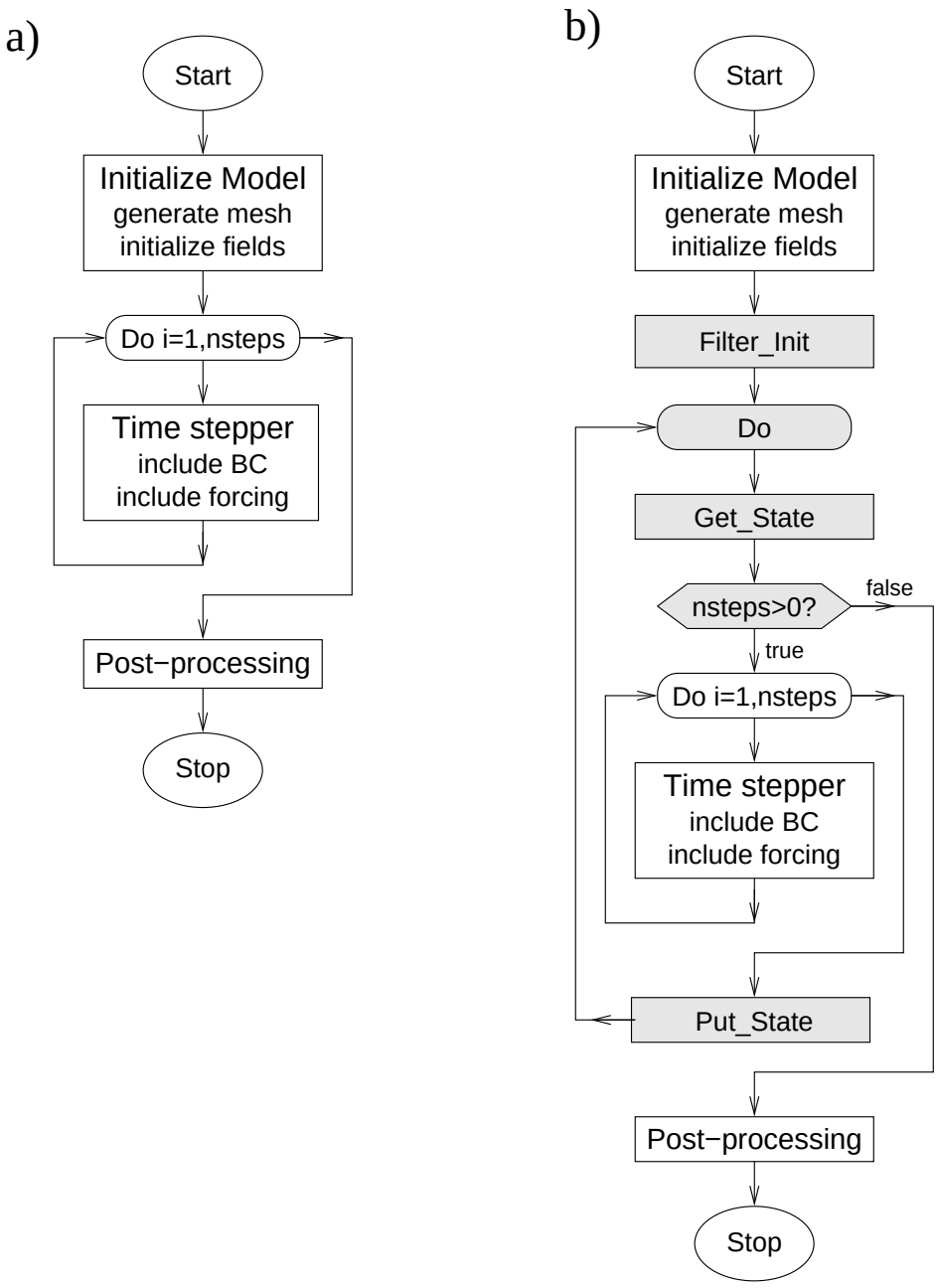


Figure 8.2: Flow diagrams: a) Sketch of the typical structure of a model performing time evolution of some physical fields. b) Structure of the data assimilation configuration of the model with attached filter. Added subroutine calls and control structures are shaded in gray.

The implementation of the filter routines has been discussed in chapter 7. The names of user supplied subroutines are handled in the framework as subroutine arguments in the filter routines and have thus to be specified in the API. This allows the user to choose the subroutine names flexibly.

8.3 Framework for Joint Process Sets for Model and Filter

First we consider the situation that the filter routines are executed by some of the processes which are used for the model evolutions. In this case, the internal variables of the filter algorithms are mainly stored using Fortran modules. With this, e.g., the ensemble matrix $\mathbf{X}$ or the counter for the ensemble member to be evolved can be shared between the different subroutines of the filter. The names of user supplied subroutines cannot be handled via modules. For this reason, the subroutine names have to be used as arguments in the call to each routine using the particular subroutines.

8.3.1 The Application Program Interface

The three subroutines *Filter_Init*, *Get_State*, and *Put_State* provide a defined interface to the filter algorithms. In addition, the user-supplied routines like the observation-related subroutines and the user analysis routine are called using a defined interface. We discuss here the interface to the three routines of the framework which are called from the model. The interfaces of the user supplied routines which are called by the filter are described in appendix B. The interfaces of these routines are equal for both process configurations. The implementation of the operations performed in these routines depend, however, on the choice whether a parallelization on basis of mode-decomposition or domain-decomposition is used.

The interface of the routine *Filter_Init* is shown as algorithm 8.1. This routine is called in the model source code by all processes. For the initialization several variables are passed to the filter. With the integer argument *type_ass* the user chooses the filter algorithm to be used. For flexibility, *subtype_ass* defines a sub-type of the filter. This might be, e.g., a variant of SEEK in which the modes in matrix $\mathbf{V}$ are not evolved [33]. The array *param_int* is a vector of variable size *dim_pint*. It holds integer parameters for the filters. In the current implementation of the filters *dim_pint* = 3 is set if the SEEK or SEIK filters are used. For EnKF, it is *dim_pint* = 4. The first entry of *param_int* holds the dimension n of the state vector. The second entry specifies the ensemble size N for EnKF or the rank r of the approximated state covariance matrix for SEEK and SEIK. The third entry specifies whether a parallelization with domain-decomposition or a decomposition over the modes of the ensemble matrix is used. For the EnKF the fourth entry is used to specify the rank of the inverse on the left hand side of equation (2.47) if a pseudo inverse has to be computed. A value of zero specifies that the solution of equation (2.47) is computed using the LAPACK routine DGESV. The array *param_real* of size *dim_preal* defines a vector of floating point parameters which

Subroutine Filter_Init(*type_ass*,*subtype_ass*,*param_int*,*dim_pint*,*param_real*, *dim_preal*,*COMM_MODEL*,*COMM_FILTER*,*COMM_COUPLE*, *modeltask*,*n_modeltasks*,*filterpe*,Init_Ensemble,*verbose*,*status*)
int *type_ass* {Type a filter algorithm, input}
int *subtype_ass* {Sub-type of filter, input}
int *param_int*(*dim_pint*) {Array of integer parameters, input}
int *dim_pint* {Size of *param_int*, input}
real *param_real*(*dim_preal*) {Array of floating point parameters, input}
int *dim_preal* {Size of *param_real*, input}
int *COMM_MODEL* {Model communicator, input}
int *COMM_FILTER* {Filter communicator, input}
int *COMM_COUPLE* {Coupling communicator, input}
int *modeltask* {Model task the process belongs to, input}
int *n_modeltasks* {Number of parallel model tasks, input}
int *filterpe* {Whether the process belongs to the filter processes, input}
external Init_Ensemble {Subroutine for ensemble initialization, input}
int *verbose* {Whether to print screen information, input}
int *status* {Output status flag of filter, output}

Algorithm 8.1: Interface to the subroutine *Filter_Init* in the case of joint process sets for model and filter.

are be required for some of the filters. For SEIK and EnKF *param_real* has a size of 1 and contains only the value of the forgetting factor ρ. For SEEK it is $dim\_preal = 2$. While the first entry of *param_real* specifies the forgetting factor ρ, the second entry sets the value of ϵ for the gradient approximation of the forecast. The flexible sizes of *param_int* and *param_real* allow for future extensions of the functionality. Next to these variables, the three communicators are handed over to the filter initialization routine. Further, the index *modeltask* of the model task of the process calling *Filter_Init* and the total number *n_modeltasks* of parallel model tasks is passed to the filter initialization routine. The argument *filterpe* specifies whether a process belongs to the filter processes. The name of the subroutine performing the ensemble generation is the next argument. The interface is completed by an argument which defines whether the filter routines will print out screen information and a final argument which serves as a status flag. It will have a non-zero value if a problem occurred during the initialization.

The subroutine *Get_State* is called in the model source code before the time stepping loop is entered. *Get_State* initializes the state fields of the model and provides the information on the current model time and the number of time steps to be computed, The interface to this routine is shown as algorithm 8.2. All parameters which are required by the filters have already been specified in the filter initialization. Accordingly, the interface of *Get_State* contains only names of subroutines and output variables which are initialized for the model time stepper. The variables *nsteps* and *time*, as well as the status flag *status* are outputs of the routine. In addition, the names of three subroutines are specified. The routines *Next_Observation* and *User_Analysis* have already

Subroutine Get_State(*nsteps*,*time*,Next_Observation,Distribute_State,
User_Analysis,*status*)
int *nsteps* {Number of time steps to be performed, output}
real *time* {Model time at begin of evolution, output}
external Next_Observation
{Subroutine to get number of time steps and current time, input}
external Distribute_State
{Subroutine to distribute state in *COMM_MODEL*, input}
external User_Analysis {Subroutine for user analysis, input}
int *status* {Output status flag of filter, output}

Algorithm 8.2: Interface to the subroutine *Get_State* in the case of joint process sets for model and filter.

been described in section 3.3.1. The routine *Distribute_State* transfers a state vector to model fields and distributes these within the model task defined by *COMM_MODEL*. In the variant with mode-decomposition, the framework itself only initializes a state vector on a single process in each model task. The model-dependent operations are then performed by the routine *Distribute_State* which is described in section 8.5.

Having computed the evolution of a model state, this forecast is stored back in the ensemble or mode matrix of the filter algorithm. This is performed in the routine *Put_State*. If *Put_State* is called after the filter forecast phase has been completed, the analysis and resampling phases are executed by this routine. In its interface, the names of several subroutines which are called by the filter analysis and resampling algorithms have to be specified. The observation-related routines *Measurement_Operator*, *Measurement*, *RinvA*, *RplusA*, and *Get_Dim_Obs* have already

Subroutine Put_State(Collect_State,Get_Dim_Obs,Measurement_Operator,
Measurement,Measurement_Ensemble,User_Analysis,RinvA,RplusA,*status*)
external Collect_State
{Subroutine to collect state vector in *COMM_MODEL*, input}
external Get_Dim_Obs
{Subroutine to provide dimension of observation vector, input}
external Measurement_Operator
{Subroutine with implementation of measurement operator, input}
external Measurement {Subroutine to initialize observation vector, input}
external Measurement_Ensemble
{Subroutine to initialize ensemble of observation vectors, input}
external User_Analysis {Subroutine for user analysis, input}
external RinvA {Subroutine for product of $\mathbf{R}^{-1}$ with some matrix, input}
external RplusA {Subroutine to add $\mathbf{R}$ to some matrix, input}
int *status* {output status flag of filter, output}

Algorithm 8.3: Interface to the subroutine *Put_State* in the case of joint process sets for model and filter.

been discussed in section 3.3.2. The routine *Measurement_Ensemble* is required in the EnKF. It provides the observation ensemble according to the observation error covariance matrix **R**. *Collect_State* performs the operation inverse to that of the routine *Distribute_State*. That is, the ensemble fields in a model task are gathered in a state vector. For mode-decomposed ensemble matrices, the state vector is gathered by a single process of this task. Next to the names of subroutines, the interface of *Put_State* contains again the status flag *status* as an output variable.

The routine *Put_State* is generic for all three filter algorithms. Due to this, the interface requires the specification of all possible subroutine names, even if they are not required for all filters. For example, SEEK and SEIK only require the routine *RinvA* but not *RplusA*. The latter routine is required by the EnKF analysis while the former one is not used by this filter. To generate an executable program all three routines must be present (possibly as an empty routine, if it is not called by the chosen filter), since they are required for the linker step. To facilitate the implementation if only one filter type is used, we have implemented specific routines like *Put_State_SEEK* for the SEEK filter. The interface of the specific put-routines contains only the names of the subroutines relevant for the chosen filter.

It would be possible to avoid the names of subroutines in the calling interfaces to *Filter_Init*, *Get_State*, and *Put_State*. This would simplify the API considerable. On the other hand this would disable the possibility to use arbitrary names for the subroutines. We prefer this flexibility, since the user is not urged to use specific names for his subroutines.

8.3.2 Process Configurations for the Filtering Framework

Before we explain the functionality of the filter interface routines and the communication of data between the filter and the model part of the data assimilation program, we discuss the configuration of the MPI communicators. These define the process topology for the data assimilation framework. In general, the data assimilation framework requires that the user initializes the communicators and provides the names of these communicators to the routine *Filter_Init*. To facilitate the initialization of the communicators, the framework includes templates for these operations. These templates can be used in most situations without changes, but can be adapted when necessary. The communicator configurations use simple 1-dimensional process topologies. Dependent on the model, it might be useful to apply other topologies inside the process sets of *COMM_MODEL*, e.g., to obtain optimal performance for 2-dimensional domain decompositions.

A possible process configuration for mode-decomposition is shown in figure 8.3. In this figure each row corresponds to the communicator which is given on the right hand side. The processes are ordered from left to right according to their logical process number which is given by the rank of the processes in the communicator *MPI_COMM_WORLD*. Thus, the entries in a single column refer to the same process. The number entries denote the rank of the process in the communicator. If no rank is

⟶ logical process number (= rank in *MPI_COMM_WORLD*)								
[0	1	2	3	4	5	6	7]	*MPI_COMM_WORLD*
[0	1]	[0	1]	[0	1]	[0	1]	*COMM_MODEL*
[0		1]		[0		1]		*COMM_COUPLE*
[0				1]				*COMM_FILTER*

Figure 8.3: Example communicator configuration for the case that the filter is executed by some of the model processes and the filter routines use a parallelization of the modes.

given for a process in the context of some communicator, this process does not attend in communications within this communicator. The brackets enclose processes which build together a process set on the communicator.

In the example the program is executed by a total of 8 processes. These are distributed into four parallel model tasks each executed by two processes in the context of *COMM_MODEL*. The filter routines are executed by two processes. These are the processes of rank 0 and 4 in the context of *MPI_COMM_WORLD*. In the context of *COMM_MODEL* the filter processes have rank 0. Each filter process is coupled to two model tasks. Thus, there are two disjoint process sets in *COMM_COUPLE* each consisting of two processes. With this configuration, the filter initialization will divide the ensemble or the mode matrix into two matrices which are stored on the two filter processes. Each matrix holds a sub-ensemble of model states. For the utilization of all four model tasks, each filter process will again distribute its sub-ensemble to the two model tasks which are coupled to it by *COMM_COUPLE*.

A simpler configuration which will be sufficient for most applications is shown in figure 8.4. Again there are four parallel model tasks each containing two processes. The filter is executed in this configuration by each process which has rank 0 in the context of *COMM_MODEL*. With this configuration, the communication scheme is simplified since no communication via *COMM_COUPLE* is required. Each process set in *COMM_COUPLE* contains only a single process and the filter processes can directly provide data to the model tasks. Using this configuration, the state or mode ensemble is distributed into sub-ensembles in the routine *Filter_Init*. In contrast to the configuration in figure 8.3, no further distribution of the ensemble is necessary.

⟶ logical process number (= rank in *MPI_COMM_WORLD*)								
[0	1	2	3	4	5	6	7]	*MPI_COMM_WORLD*
[0	1]	[0	1]	[0	1]	[0	1]	*COMM_MODEL*
[0]		[0]		[0]		[0]		*COMM_COUPLE*
[0		1		2		3]		*COMM_FILTER*

Figure 8.4: Example communicator configuration analogous to that in figure 8.3. Here the filter is executed by all processes which have rank 0 in *COMM_MODEL*.

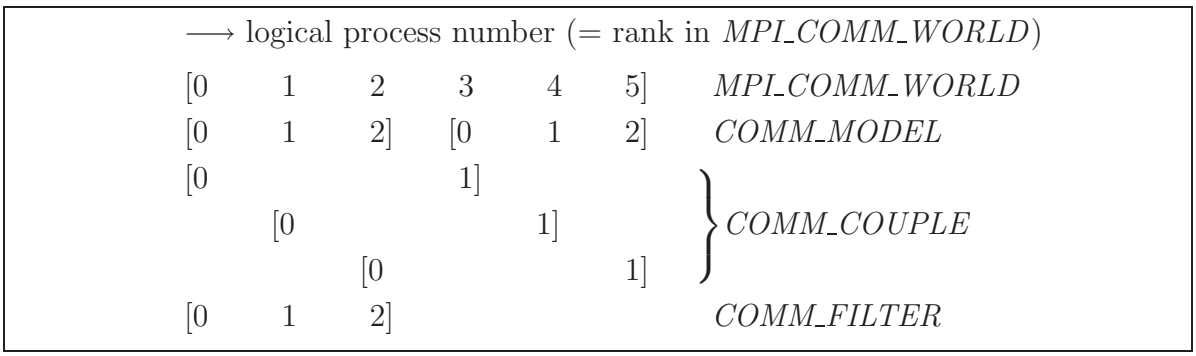


Figure 8.5: Example communicator configuration for the case of domain-decomposed states. The filter is executed by some of the model processes.

If a domain-decomposition is used for the parallelization of the model and the filter parts of the program, the configuration of the processes is distinct from the case of mode-decomposition. considered is the situation that the filter uses the same domain-decomposition of the states as the model. Figure 8.5 shows a possible process configuration. Here the program is executed by six processes in total. These are distributed into two model tasks each consisting of three processes. The filter routines are executed by all processes of one of the model tasks. Hence, the sub-state from this model task can be directly transfered between the local ensemble matrix and the model fields. The second model task is connected to the filter via *COMM_COUPLE*. With domain-decomposition, the initialization of the sub-states is performed in the initialization phase of the filter. The filter operates on the whole ensemble of local sub-states. To use multiple model tasks the ensemble is distributed into sub-ensembles. These are sent to the model tasks via *COMM_COUPLE*.

A simplified configuration is possible which uses only a single domain-decomposed model task. This would lead to a trivial coupling communicator which consists of process sets containing a single process each. Thus, no communication in *COMM_COUPLE* would be necessary and the overall MPI communication would be minimized. This configuration would, however, require a model with very efficient parallelization. On the other hand, overall scalability would be limited, since only a single model evolution is computed at a time.

8.3.3 The Functionality of the Framework Routines

To gain further insight in the functionality of the data assimilation framework, we discuss here the operations which are performed in its main routines. The filter algorithms are hidden behind the three subroutines *Filter_Init*, *Get_State*, and *Put_State*. Due to this, the filter main routine, which was discussed in section 3.3.1, is split into two parts. These parts reside in *Get_State* and *Put_State*. Some additional operations are contained in these routines which are required for the parallel execution of the data assimilation framework.

The interface to the routine *Filter_Init* has been shown in algorithm 8.1. Algorithm 8.4 sketches the operations which are performed in this routine when the

SEIK filter is used with a mode-decomposed ensemble matrix. The routine is called by all processes. Here several parameters are initialized, like the chosen filter algorithm or the ensemble size. These parameters are shared between the filter routines using Fortran modules. All subsequent operations in *Filter_Init* are only performed by the filter processes. First, the sizes of sub-ensembles are computed. Subsequently, the arrays required for the filter are allocated. These are the state vector $\mathbf{x}$ and the local ensemble matrices $\mathbf{X_p}$. In addition, the full ensemble matrix $\mathbf{X}$ is allocated on the filter process with rank 0. After the allocation of the fields, the user-supplied subroutine *Init_Ensemble* is called. For the SEIK filter, this routine initializes the ensemble matrix. If a parallelization with mode-decomposition is used, *Init_Ensemble* is only called by the process with rank 0. Here the full ensemble matrix is initialized. Subsequently, it is necessary to distribute sub-ensembles to all filter processes. This is performed by MPI communication operations.

In the case of domain-decomposed states, the routine *Init_Ensemble* is called by all filter processes. The routine has to provide the full state ensemble for the local domain of each process. Since the state ensembles are readily initialized by all filter processes no further distribution of the ensembles is performed in *Filter_Init*. A similar technique could be used for a mode-decomposed ensemble matrix. That is, *Init_Ensemble* is called by each filter process with the local sub-ensemble as argument. Then *Init_Ensemble* initializes only this local sub-ensemble. Since the sub-ensembles are readily initialized on the filter processes, no distribution of sub-ensembles would be required in *Filter_Init*. Using this variant would avoid the storage of the full ensemble matrix on a single process. On the other hand the user would be obliged to implement *Init_Ensemble* such that all sub-ensembles are initialized correctly. From this point of view, the first variant, which initializes the full ensemble matrix on a single process, is simpler to use. If memory limitations render the allocation of the full ensemble matrix on a single process impossible, the initialization should directly operate on the sub-ensembles. To allow for this flexibility, *Filter_Init* contains both variants.

The subroutine *Get_State* is called prior to each model state evolution. Its structure is sketched in algorithm 8.5 for the SEIK and EnKF filters. If the routine is called for the very first time, it calls the user analysis routine *User_Analysis*. This permits to analyze the initial ensemble consistently with the calls to *User_Analysis* which are performed during the assimilation. Also the ensemble counter *member* is set to one at the very first call to *Get_State*. For the remainder of the routine, this signals that a new forecast phase has to be performed.

If *Get_State* is called in the beginning of a forecast phase (i.e., with $member = 1$), the routine *Next_Observation* is called by the process of rank 0 in *COMM_FILTER*. *Next_Observation* initializes the number of time steps *nsteps* for the next forecast phase and the current model time *time*. Subsequently, the value of *nsteps* is distributed to all processes. If $nsteps > 0$, also the variable *time* is distributed to all processes by a broadcast operation. If the number of filter processes is smaller than the number of model tasks, as was the case in figure 8.3, the sub-ensemble of each filter process is further distributed such that each model task holds several ensemble members. This concludes the initialization of a forecast phase.

```
Subroutine Filter_Init(...)
    ⋮
    int mype_filter  {Rank of process in COMM_FILTER}
    int npes_filter  {Number of processes in COMM_FILTER}
    ⋮
 1: initialize parameters
 2: if filterpe == 1 then
 3:    initialize local ensemble sizes N_p
 4:    allocate fields: X_p(n, N_p), x(n)
 5:    if mype_filter == 0 then
 6:       allocate ensemble matrix X(n, N)
 7:       call Init_Ensemble(X)  {Initialize full ensemble matrix}
 8:       for i = 1, npes_filter do
 9:          send sub-ensemble X(j_p : j_p + r_p - 1) to filter process i  {With MPI_Send}
10:       end for
11:       deallocate field X
12:    else if mype_filter > 0 then
13:       receive sub-ensemble X_p  {With MPI operation MPI_Recv}
14:    end if
15: end if
```

Algorithm 8.4: Sketch of the operations which are executed in the routine *Filter_Init* for the case of mode-decomposition. The interface to this routine is shown as algorithm 8.1

When *Get_State* is called during a forecast phase, it calls the user-supplied routine *Distribute_State*. Here the model fields are initialized from the state vector which is provided to *Distribute_State* as a subroutine argument. Since the state vector is only initialized on a single process of a model task, it might also be necessary to distribute the state information to the other processes of the model task.

Distribute_State is not called directly by the model routines. Accordingly, the model fields or information on the model grid cannot be supplied as subroutine arguments. Thus, *Distribute_State* requires that the model fields are available via Fortran modules or 'common' blocks. We will discuss this issue in section 8.5.

The routine *Put_State* is called after a model state has been evolved by the model time stepper. Algorithm 8.6 sketches the operations which are performed in this routine for the SEIK filter. During the forecast, the user-supplied routine *Collect_State* is called with the current ensemble state vector as argument. Also the ensemble counter *member* is incremented. *Collect_State* initializes the forecasted state vector from the evolved model fields. This is the inverse operation to that performed by *Distribute_State*. We will discuss *Collect_State* in section 8.5.

If the forecast of all ensemble members is not yet finished, the program exits *Put_State* and loops back to *Get_State* in order to evolve the next ensemble member. If the ensemble forecast is completed, the filter processes proceed in routine *Put_State*

```
Subroutine Get_State(...)
    ⋮
    int firsttime {Flag whether routine is called the very first time}
    int member {ensemble counter; shared using Fortran module}
    ⋮
 1: if firsttime == 1 then
 2:    call User_Analysis(...)
 3:    firsttime ← 0
 4:    member ← 1
 5: end if
 6: if member == 1 then
 7:    if mype_filter == 0 then
 8:       call Next_Observation(step,nsteps,time) {User supplied routine}
 9:    end if
10:    broadcast nsteps to all processes {With operation MPI_Bcast}
11:    if nsteps > 0 then
12:       broadcast time to all processes {With operation MPI_Bcast}
13:       distribute sub-ensembles {With operations MPI_Send and MPI_Recv}
14:    end if
15: end if
16: if nsteps > 0 then
17:    call Distribute_State(n, X_p(:, member)) {User supplied routine}
18: end if
```

Algorithm 8.5: Sketch of the operations which are executed in the routine *Get_State*. The interface to this routine is shown as algorithm 8.2

to perform the analysis and resampling phases of the filter algorithm. If there are less filter processes than model tasks, all ensemble members are gathered by the filter processes. Consecutively, the filter update phases are performed by calling *SEIK_Analysis* and *SEIK_Resample* and the user supplied analysis routine *User_Analysis*. After the update, the ensemble counter *member* is reset to one and the filter processes exit *Put_State*. Only the filter processes perform the update. The remaining processes reset the ensemble counter and proceed directly to the routine *Get_State*. Here, they wait to receive the variable *nsteps* which is send from the filter process with rank 0 in *COMM_FILTER* to all processes by a broadcast operation (line 10 of algorithm 8.5).

8.4 Framework for Model and Filter on Disjoint Process Sets

The variant of executing the model and the filter parts of the data assimilation program on disjoint process sets permits a very clear separation between these to parts of the program. All processes will call the filter initialization routine. Then, the filter

```
Subroutine Put_State(...)
    ⋮
    int member  {ensemble counter; shared using Fortran module}
    int N_p  {local ensemble size; shared using Fortran module}
    ⋮
 1: call Collect_State(n, X_p(:, member))
 2: member ← member + 1
 3: if member = N_p + 1 then
 4:    gather sub-ensembles  {With operations MPI_Send and MPI_Recv}
 5:    if filterpe == 1 then
 6:       call User_Analysis(...)  {User supplied routine}
 7:       call SEIK_Analysis(...)  {Perform filter analysis}
 8:       call SEIK_Resample(...)  {Perform resampling}
 9:       call User_Analysis(...)  {User supplied routine}
10:    end if
11:    member ← 1
12: end if
```

Algorithm 8.6: Sketch of the operations which are executed in the routine *Put_State*. The interface to this routine is shown as algorithm 8.3

processes proceed directly to the filter main routine. The model processes will exit the initialization routine and proceed to the model time stepper loop. During the data assimilation phase, the model and filter parts of the program are connected only by MPI communication.

8.4.1 The Application Program Interface

The application program interface in the case of disjoint process sets for model and filter consists again of the three routines *Filter_Init*, *Get_State*, and *Put_State*. In addition, the observation-related subroutines and the routines *Distribute_State* and *Collect_State* are required. These routines can be identical to those routines which are used in the framework discussed in section 8.3.1. Finally, the user analysis routine *User_Analysis* is required. The interface for this routine is identical to that of the joint-process case.

The interface of *Filter_Init* is shown as algorithm 8.7. It is called by all processes, to allow also for the initialization of parameters for the routines *Get_State* and *Put_State* which will only be executed by the model processes. The required parameters in the interface of *Filter_Init* are the same as in the case of joint process sets for model and filter. These parameters have been documented in section 8.3.1. Also the name of the subroutine performing the ensemble initialization has to be provided. In the call to *Filter_Init* the API for disjoint process sets requires, in addition, the specification of the observation-related subroutines and the user analysis routine. This is necessary since the filter processes directly call the main filter routine in *Filter_Init*.

Subroutine Filter_Init(*type_ass*,*subtype_ass*,*param_int*,*dim_pint*,*param_real*,
dim_preal,*COMM_MODEL*,*COMM_FILTER*,*COMM_COUPLE*,
filterpe,Init_Ensemble,Get_Dim_Obs,Next_Observation,Measurement_Operator,
Measurement,Measurement_Ensemble,User_Analysis,RinvA,RplusA,
verbose,*status*)
int *type_ass* {Type a filter algorithm, input}
int *subtype_ass* {Sub-type of filter, input}
int *param_int*(*dim_pint*) {Array of integer parameters, input}
int *dim_pint* {Size of *param_int*, input}
real *param_real*(*dim_preal*) {Array of floating point parameters, input}
int *dim_preal* {Size of *param_real*, input}
int *COMM_MODEL* {Model communicator, input}
int *COMM_FILTER* {Filter communicator, input}
int *COMM_COUPLE* {Coupling communicator, input}
int *modeltask* {Model task the process belongs to, input}
int *n_modeltasks* {Number of parallel model tasks, input}
int *filterpe* {Whether the process is a filter process, input}
external Init_Ensemble {Subroutine for ensemble initialization, input}
external Get_Dim_Obs
{Subroutine to provide dimension of observation vector, input}
external Next_Observation
{Subroutine to get number of time steps and current time, input}
external Measurement_Operator
{Subroutine with implementation of measurement operator, input}
external Measurement {Subroutine to initialize observation vector, input}
external Measurement_Ensemble
{Subroutine to initialize ensemble of observation vectors, input}
external User_Analysis {Subroutine for user analysis, input}
external RinvA {Subroutine for product of $\mathbf{R}^{-1}$ with some matrix, input}
external RplusA {Subroutine to add $\mathbf{R}$ to some matrix, input}
int *verbose* {Whether to print screen information, input}
int *status* {Output status flag of filter, output}

Algorithm 8.7: Interface to the subroutine *Filter_Init* in the case of disjoint process sets for model and filter.

Filter_Init is generic for all three filter algorithms. As for the routine routine *Put_State* in the case of joint processes in section 8.3.1, all subroutine names have to be specified in the interface, even if they are not required for all filters algorithms. To facilitate the implementation, the framework also provides specific initialization routines for the filters. These routines require only the specification of the subroutines which are used for the particular filter.

Algorithms 8.8 and 8.9 show respectively the routines *Get_State* and *Put_State*. As these routines are called from the model routine, they are only executed by the model processes. The routines receive and send the state vectors. Furthermore, *Get_State* receives the time stepping information. In addition, both routines control the transition between the state vector and the model fields. Direct outputs of *Get_State* are again the number of time steps (*nsteps*) and the model time at begin of the evolution (*time*). Next to these variables and the status flag *status*, only the subroutine *Distribute_State* has to be specified. The functionality of *Distribute_State* is the same as in the case of joint processes for model and filter. The interface of *Put_State* is considerably simpler here than in the configuration with joint processes. Only the subroutine *Collect_State* has to be specified since the update routines of the filter are not directly called by *Put_State*. The status flag is given as the second argument of the interface.

8.4.2 Process Configurations for the Filtering Framework

A possible process configuration for mode-decomposed ensemble matrices is shown in figure 8.6. The program is executed by six processes. There are two model tasks which are executed by two processes each. The remaining two processes are used to execute the filter. Each filter process is coupled to one model task by the communicator *COMM_COUPLE*. Here, the communication in *COMM_COUPLE* is always necessary, since it couples the disjoint process sets of filter and model. During the forecast phase each filter process sends the states of its sub-ensemble to the model task connected to it and receives forecasted state vectors. The model evaluations are performed only by the model processes while the filter processes wait for data. The filter analysis and resampling are computed only by the two filter processes. Meanwhile, the model processes idle.

Figure 8.7 shows a possible configuration for domain-decomposed states. As before, six processes are used in total. Two processes are again used for the filter. The forecasts are evaluated on two model tasks, each consisting of two processes. The communicator *COMM_COUPLE* now couples each filter process with respectively one process of both model tasks. Thus, during the forecast phase, a filter process sends local state vectors to both model tasks. When all processes of a model task have received a sub-state, they start with the model evaluations.

```
Subroutine Get_State(nsteps,time,Distribute_State,status)
    int nsteps {Number of time steps to be performed, output}
    real time {Physical time at begin of evolution, output}
    external Distribute_State
        {Subroutine to distribute state in COMM_MODEL, input}
    int status {Output status flag of filter, output}
    int n {Model state dimension}
    real x(n) {State vector}
    int mype_model {Process rank in COMM_MODEL}

 1:   if mype_model == 0 then
 2:     receive nsteps in COMM_COUPLE {With operation MPI_Recv}
 3:   end if
 4:   broadcast nsteps in COMM_MODEL {With operation MPI_Bcast}
 5:   if nsteps > 0 then
 6:     if mype_model == 0 then
 7:       receive time in COMM_COUPLE {With operation MPI_Recv}
 8:       receive x in COMM_COUPLE {With operation MPI_Recv}
 9:     end if
10:     broadcast time in COMM_MODEL {With operation MPI_Bcast}
11:     call Distribute_State(n,x)
12:   end if
```

Algorithm 8.8: Pseudo code of the subroutine *Get_State* in the case of disjoint process sets for model and filter.

```
Subroutine Put_State(Collect_State,status)
    external Collect_State
        {Subroutine to collect state vector in COMM_MODEL, input}
    int status {output status flag of filter, output}
    int n {Model state dimension}
    real x(n) {State vector}
    int mype_model {Process rank in COMM_MODEL}

 1:   call Collect_State(n,x)
 2:   if mype_model == 0 then
 3:     send x in COMM_COUPLE {With operation MPI_Send}
 4:   end if
```

Algorithm 8.9: Pseudo code of the subroutine *Put_State* in the case of disjoint process sets for model and filter.

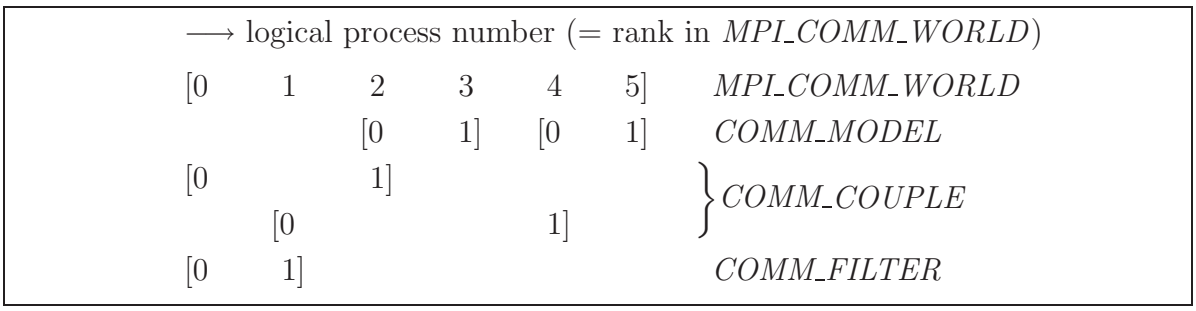


Figure 8.6: Example communicator configuration for the case that model and filter are executed by disjoint process sets and the filter routines use a parallelization over the modes of the ensemble matrix.

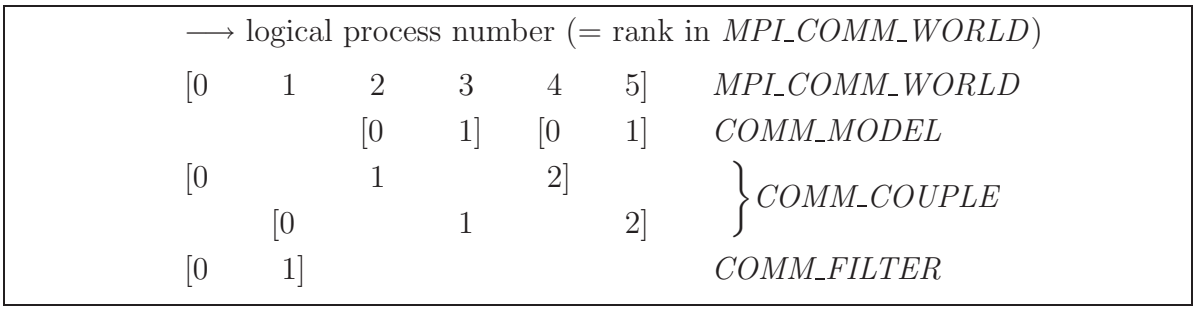


Figure 8.7: Example communicator configuration for the case of domain-decomposed states and execution of model and filter parts by disjoint process sets. The example is analogous to that in figure 8.6. In contrast to the mode-decomposed case, each filter process is coupled to respectively one process of both model tasks.

8.4.3 Execution Structure of the Framework

The data assimilation for disjoint process sets for model and filter exhibits a clear separation between the model and filter parts. Both are executed concurrently on their respective processes. A flow diagram for the framework which exemplifies the SEIK filter is shown in figure 8.8. The thick green lines symbolize communication.

On execution of the program, the MPI communicators are initialized by all processes in global operations. Since in this phase of the program all processes are available, the user has to take care that the subsequent model initialization is performed only by the model processes. The allocation and initialization of model fields is not required by the filter processes. After the model initialization, the filter initialization routine *Filter_Init* is called by all processes. In this routine, the model processes store the information on the communicators *COMM_MODEL* and *COMM_COUPLE* while the filter processes store the information on *COMM_COUPLE* and *COMM_FILTER*. Subsequently, the model processes exit the filter initialization routine. The filter processes proceed in *Filter_Init* by allocating the arrays which are required for the chosen filter. Then the state vector $\mathbf{x}$ and the ensemble matrix $\mathbf{X}$ or the mode matrix $\mathbf{V}$ are initialized and sub-ensembles are distributed to all filter processes. Finally the filter processes call the filter main routine whose components are shown on the right hand side of figure 8.8.

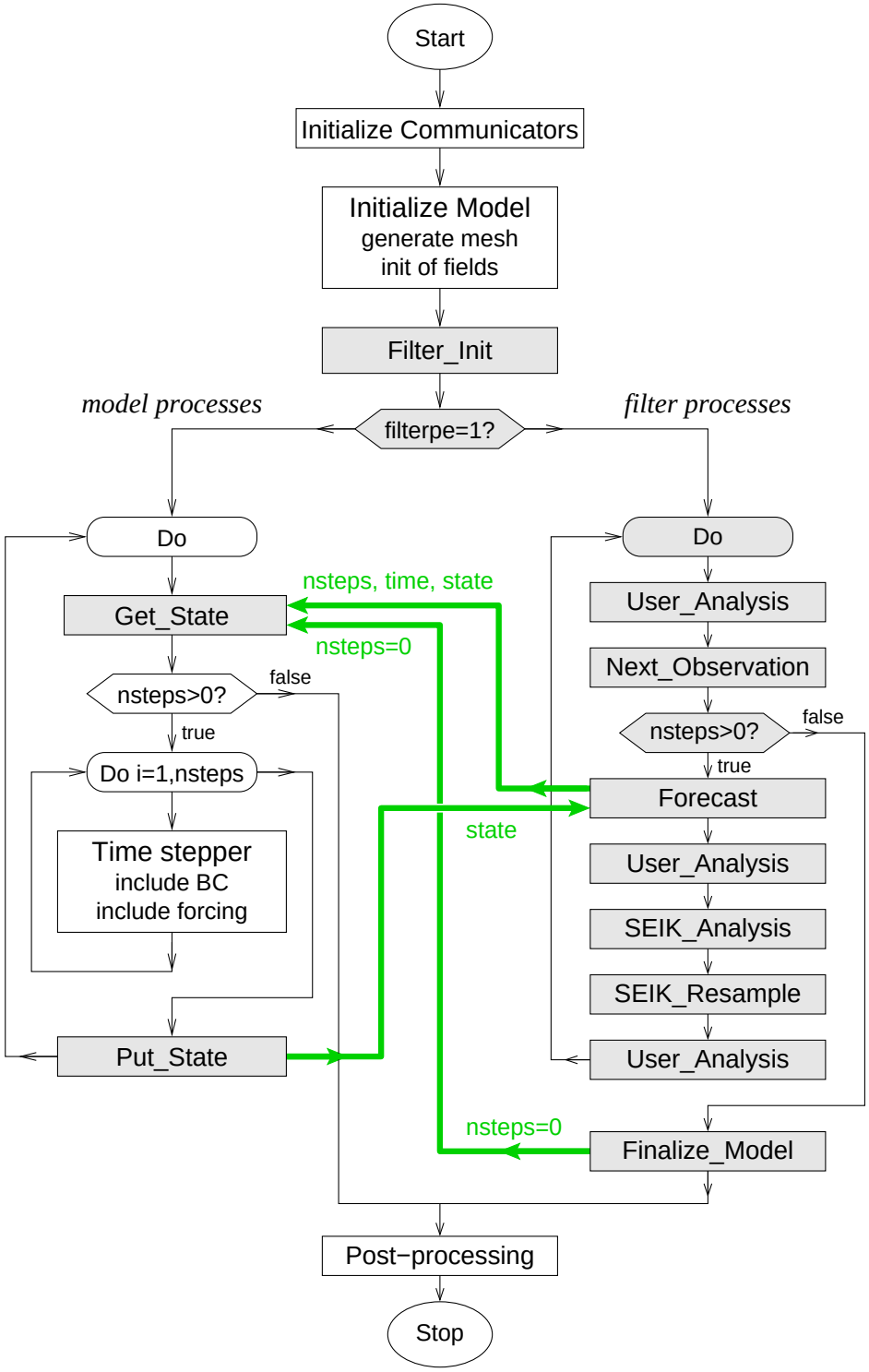


Figure 8.8: Flow diagram for the framework when filter and model are executed by disjoint process sets. Exemplified is the program flow for the SEIK filter. Shaded in gray are the routines of the filter framework. The thick green lines denote communication. The parts of the program which are horizontally centered are executed by all processes. After the initialization, the program splits into the model part displayed on the left hand side and the filter part on the right hand side. Both parts are connected by communication operations.

Having left the routine *Filter_Init*, the model processes proceed to the forecast loop shown on the left hand side of figure 8.8. In *Get_State* (see algorithm 8.8), the processes wait to receive the value of *nsteps* which is sent by the filter. If $nsteps = 0$, no forecast has to be performed. Thus, no further operations are necessary in *Get_State* and the forecast loop is exited. If $nsteps > 0$, the processes also receive the variable *time* and the state vector $\mathbf{x}$ to be evolved. Subsequently, the routine *Distribute_State* is called which initializes the model fields on the basis of the state vector $\mathbf{x}$. Then the evolution of the state is performed by the model time stepper. After the evolution, the routine *Put_State* is called. This routine is shown as algorithm 8.9. Here *Collect_State* is called to initialize the forecasted state vector from the model fields on the model process with rank 0. Subsequently, this process sends the state vector $\mathbf{x}$ to the filter. This completes the forecast loop and the processes return to the begin of the unconditioned loop.

The structure of the filter main routine on the right hand side of figure 8.8 is essentially the same as that of the serial algorithm which we have discussed as algorithm 3.1. An addition to this algorithm is the subroutine *Finalize_Model*. It is required in the parallel program to send *nsteps* with a value of zero to the model tasks. As discussed above, this signalizes to the model tasks to exit the forecast loop.

The subroutine *Forecast* controls the loop over all ensemble members to be evolved. It is shown as algorithm 8.10. In the configuration with disjoint processes for filter and model, an algorithm is used which sends a only single ensemble state vector to the available model tasks. The filter part of the algorithm uses non-blocking MPI operations. These only post the communication operation and immediately return from the function even if the communication operation is not yet completed. In contrast to this, the routines *Get_State* and *Put_State* apply blocking MPI operations to ensure that the data has been received or send completely. Sending and receiving single state vectors permits a flexible handling of the forecast phase. If a forecasted state vector is received back from some model task, a new ensemble state vector can be send immediately to this task if there are any ensemble states left. For sufficiently large ensembles, this ensures a good load balancing since faster model tasks can evolve more ensemble states than slower model tasks. This algorithm is more flexible than the configuration used for joint process sets for filter and model. There the sizes of sub-ensembles are set during the initialization phase of the framework. In addition, the memory requirements are smaller here. In the case of mode-decomposition, a single state vector is allocated on the model processes with rank 0 in *COMM_MODEL*. No filter-related memory allocations are required on the remaining model processes. For domain-decomposition a single sub-state is allocated on each model process. For the configuration using joint process sets for filter and model, it is required to allocate sub-ensembles of state vectors.

```
Subroutine Forecast(step,nsteps,time)
    int step {Current time step, input}
    int nsteps {Number of time steps to be computed, output}
    real time {Current model time, output}
    int n {Model state dimension}
    int N_p {Size of local state ensemble}
    real X_p(n, N_p) {Local state ensemble}
    int npes {Number of processes in COMM_COUPLE}
    int status(npes − 1) {Status array; idle: 0, working: 1}
    int send_ens {Counter for ensemble member to become evolved}
    int get_ens {Number of received state vectors}

 1:   status(1 : npes − 1) ← 0 {Set status to idle for all tasks}
 2:   send_ens ← 1 {Send first ensemble member}
 3:   get_ens ← 0 {No state received yet}

 4:   loop
 5:     for task = 1, npes − 1 do
 6:       if status(task) == 1 then
 7:         Test whether receiving from task has been completed
                                        {With operation MPI_Test}
 8:         if receiving of task completed then
 9:           get_ens ← get_ens + 1 {Increase counter of received states}
10:           status(task) ← 0 {Set task to idle}
11:         end if
12:       end if
13:       if status(task) == 0 then
14:         send nsteps to task {With operation MPI_ISend}
15:         send time to task {With operation MPI_ISend}
16:         send X_p(:, send_ens) to task {With operation MPI_ISend}
17:         post receiving of X_p(:, send_ens) from task {Operation MPI_IRecv}
18:         send_ens ← send_ens + 1 {Increase index of member to send}
19:         status(task) ← 1 {Set task to working}
20:       end if
21:     end for
22:     if get_ens == N_p then
23:       Exit loop
24:     end if
25:   end loop
```

Algorithm 8.10: Structure of the routine of the filter framework which controls the ensemble forecast in the case of SEIK and EnKF. (For SEEK, the state estimate itself is also evolved. Hence, the forecast routine for SEEK contains an extension for evolving the state estimate.) The used MPI operations are non-blocking. Thus, the algorithm directly proceeds after posting a MPI_ISend or MPI_IRecv operation.

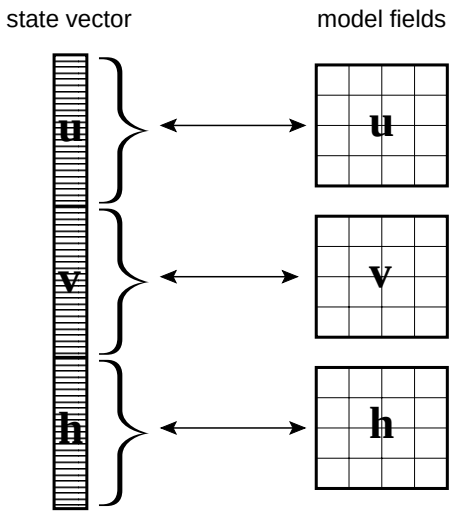


Figure 8.9: Transition between the abstract state vector (left hand side) and the model fields (right hand side). Shown is an example of three model fields of equal sizes. This example originates from the experiments with the shallow water model discussed in chapter 4. **u** and **v** are the two horizontal velocity components while **h** is the surface elevation.

8.5 Transition between the State Vector and Model Fields

The filter algorithms operate solely on the abstract state vectors. All operations which require information on the physical nature of an element of the state vector are performed in user-supplied routines. The arrangement of elements in the state vector is defined in the initialization routine *Init_Ensemble*. Here the user choses how to order the information on different physical quantities and from different physical locations. The observation-dependent routines have to consider this ordering to allow for a consistent implementation, e.g., of the measurement operator or the initialization of the observation vector. The arrangement of the elements in the state vector is also important in the routines *Distribute_State* and *Collect_State*. These routines are executed by all model processes. In contrast to this, the other user-supplied routines, are executed only by the filter processes. Figure 8.9 exemplifies the transition between the abstract state vector and model fields for the experiment using the shallow water equations which has been considered in chapter 4. The model consists of three fields, namely, the two velocity components **u**, **v** and the sea surface elevation **h**. Each of these fields is 2-dimensional. For the filter, the model fields are stored successively in the 1-dimensional state vector.

The routine *Distribute_State* is shown as algorithm 8.11. It is called from the routine *Get_State*. The purpose of *Distribute_State* is to initialize the model fields from the state vector such that the state information is sufficiently initialized for the model time stepper.

Subroutine Distribute_State(n,$\mathbf{x}$)
int n {State dimension, input}
int $\mathbf{x}(n)$ {State vector to be distributed, input}

... Initialize and distribute model fields ...

Algorithm 8.11: Interface of the subroutine *Distribute_State* which performs the transition from the state vector of the filter and the model fields.

If *Distribute_State* is called in the case of a mode-decomposed ensemble matrix, a full state vector $\mathbf{x}$ of dimension n is initialized by a single process of the model task. If the model task consists of a single process, the model fields can be directly initialized, e.g., by copying the data into the model fields. If the model task consists of multiple processes, the required operations depend on the type of the parallelization. For example, the finite element model which will be used in the experiments in chapter 9 requires that the model fields are fully initialized on all processes. Thus, the model fields are first initialized in *Distribute_State* on the process which holds the state vector. Subsequently, the model fields are distributed to the other processes in the model task by MPI operations.

If *Distribute_State* is called in the case of domain-decomposed states, each model process holds that part $\mathbf{x_p}$ of the state vector which corresponds to its local domain. Hence, *Distribute_State* will perform only the initialization of the model fields in the local domain. As long as the domain-decomposition of model and filter coincide, no communication operations are necessary.

The routine *Collect_State* is shown as algorithm 8.12. It performs the inverse operations to those of *Distribute_State*. If domain-decomposition is used, the local state vector is initialized on each model process. For mode-decomposition, the state vector, which is allocated on one of the model processes, is initialized using the evolved model fields. If the state information is distributed over the model processes, it is necessary to gather them with communication operations on the process holding the state vector. With the finite element model used in chapter 9, the evolved model fields are fully initialized on all processes of the model task. Hence, no communication operations are required.

Subroutine Collect_State(n,$\mathbf{x}$)
int n {State dimension, input}
int $\mathbf{x}(n)$ {State vector to be distributed, input}

... Initialize state vector from model fields ...

Algorithm 8.12: Interface to the subroutine *Collect_State* which initializes a state vector from the model fields.

A particular issue of the routines *Distribute_State* and *Collect_State* is that they are not directly called by the model routines. This structure of the interface permits to hide these filter-related operations from the model. It has, however, the drawback

Subroutine Get_State_Alt(*nsteps*,*time*,*n*,**x**,*status*)
int *nsteps* {Number of time steps to be performed, output}
real *time* {Physical time at begin of evolution, output}
int *n* {Model state dimension,input}
real **x**(*n*) {State vector,output}
int *status* {Output status flag of filter, output}

Algorithm 8.13: Alternative interface of the subroutine *Get_State* in the case of disjoint process sets for model and filter. The initialization of model fields is not performed in the subroutine, but the state vector **x** is an argument of the interface. This permits to initialize the model fields directly in the model routines.

that model-specific variables and arrays cannot be used as subroutine arguments. In particular, the arrays holding model fields and variables with specifications of the model grid cannot be provided as subroutine arguments. Hence, it is necessary to use Fortran modules or common blocks to provide the routines *Distribute_State* and *Collect_State* with model fields and specifications of the model grid. For models fulfilling these implementation issues, the framework can be used with the clear separation between model and filter. If, however, a model does not support this type of storage, an alternative implementation of the routines *Get_State* and *Put_State* is necessary.

Algorithm 8.13 shows the alternative variant of *Get_State* for the configuration using disjoint process sets for model and filter. The algorithm is comparable with the original implementation shown as algorithm 8.8. The routine *Distribute_State* is not called in the alternative implementation. In addition, the interface is changed to include the state dimension n and an array $\mathbf{x}(n)$ for the state vector. This array has to be allocated in the model source code. In *Get_State_Alt*, the state vector **x** is initialized on a single process if mode-decomposition is used. For domain-decomposition, a sub-state for the local domain is initialized on all processes. Since the state vectors are known in the model context in this alternative implementation, it is possible to initialize the model fields without using Fortran modules or common blocks.

8.6 Summary and Conclusions

A framework for parallel data assimilation based on Kalman filter methods was introduced. The framework is based on a clear separation between the model, the filter, and the observational parts. This allows for a structure which requires only minimal changes in an existing model source code when a data assimilation system is implemented using the filter framework. With the framework, an application program interface was introduced which defines the calling structure of the framework routines which are called by the model. Also the interfaces to user-supplied routines are defined. These are, e.g., routines which are related to the observations or routines to transfer the state vectors used in the filter algorithms to model fields and vice versa. The interface permits to switch easily between different filter algorithms. In addition, changes to the model and filter source codes can be conducted independently.

Table 8.1: Advantages (+) and drawbacks (−) of the frameworks for the two different process configurations.

one process set for filter and model	disjoint process sets
− allocation of sub-ensemble on one process of each model task	+ allocation of a single state vector on one process of each model task
− allocation of filter fields on those model processes which are also filter processes	+ allocation of filter fields on processes separate from the model processes
+ no additional processes required for the filter part	− processes additional to the model processes are necessary for the filter part
+ reduced amount of communication if the number of model tasks equals the number of filter processes	− high amount of communication, since each model state vector has to be communicated between filter and model processes
+ model grid information allocated also on filter processes	− model grid information not allocated on filter processes
− load balancing of the forecast by a priori specification of sub-ensemble sizes	+ flexible load balancing due to communication of single model state vectors
− inflexible possibilities of process configurations to achieve good load balance	+ flexible choice of process configurations; model and filter can even be executed on different computers

The framework was introduced for two different process configurations. The filter can either execute by some of the model processes (which is denoted below as joint process sets) or the filter and model parts are executed by disjoint process sets. Both variants permit to handle domain-decomposed state vectors as well as a parallelization which decomposes of the ensemble or mode matrices over the modes. To compare the two different process configurations of the framework, advantages and drawbacks of the two configurations are summarized in table 8.1.

A major drawback of the configuration using joint process sets is that at least a part of the ensemble or model matrix has to be allocated on one process of each model task. This can considerably increase the memory requirements of these processes, which also hold fields needed by the model. In addition, fields which are required for the analysis and resampling phases of the filters are allocated on those processes which are also filter processes. These memory requirements can be critical if the computer used for the data assimilation computations poses strong memory limitations. The issue of memory requirements is minor for the case of disjoint process sets. Here only a single state vector is allocated on a single process of each model task. The fields which are required for the filter operations are allocated on the filter processes which are separated from the model processes.

An advantage of the configuration using joint process sets is that the execution of the filter does not require additional processes. All processes of the program are used for model evaluations. In contrast to this, additional processes for the filter part of the program, besides the processes performing the model evaluations, are required for the configuration using disjoint process sets. During the forecast phase, these processes only send control information for the forecast, and communicate state vectors. For large-scale ocean models, the forecast of a state vector takes significantly longer than the communication between the filter and model processes. Due to this, the filter processes will idle most of the time.

Besides the requirement of additional processes for the filter, the configuration with disjoint process sets communicates more data than the variant using joint process sets. This is due to the fact that all ensemble state vectors, which have to be evolved, need to be send from the filter processes to the model processes and vice versa. For a parallelization using mode-decomposed matrices, the least amount of communication is required in the case of joint process sets if the number of filter processes equals the number of model tasks. In this situation, a sub-ensemble is allocated on each filter process. The communication reduces to that amount which is necessary to distribute the state information to all processes in a model task. For domain-decomposed states, the amount of communications between filter and model can be reduced to zero if the configuration of joint process sets and a single model task is used.

A further potential advantage of the configuration using joint process sets lies in the fact that the information on the model grid is also allocated on the filter processes. This can be beneficial, e.g., for the implementation of the measurement operator if it requires information on the spatial positions of observations and the elements of the state vector. In the case of disjoint process sets, this information has to be initialized separately from the model.

In addition to reduced memory requirements, the configuration using disjoint process sets is significantly more flexible in the configuration of the MPI communicators. Since only single model states are communicated between filter and model tasks, possible deviations in the speed of different model tasks are easily balanced by evolving more states with the faster model tasks than with the slower ones. This flexibility cannot be achieved with joint process sets. Due to the strong separation of filter and model, the configuration using disjoint process sets even permits to execute the filter part of the program on a different computer than the model tasks. Also it is possible to execute model tasks on different computers or to compute forecasts concurrently using different models.

Concluding, this comparison showed, that neither the configuration with joint process sets nor the configuration using disjoint process sets for the filter and model parts of the program is clearly preferable. The variant with joint process sets should be preferred if the computer memory permits to store sub-ensembles as well as the fields required for the filter analysis and resampling algorithms on the same processes as the model fields. Joint process sets permit to use all available processes for the model evaluations and reduces the amount of communicated data. If it is not possible to store the filter fields on the same processes as the model fields, the variant using disjoint process sets for filter and model is preferred. This variant should also be chosen if the use of multiple computers is desired to solve the data assimilation problem.

Chapter 9

Filtering Performance and Parallel Efficiency

9.1 Introduction

The parallel filtering framework developed in the preceding chapter 8 has been implemented with the Finite Element Ocean Model (FEOM) [12]. The implementation also includes the parallelized filter algorithms developed in chapter 7. FEOM is parallelized using MPI. Mainly the solver step, required for the implicit time stepping scheme of FEOM, is performed in parallel. The model state fields have to be fully allocated and initialized by all model processes.

The data assimilation system, which is obtained by combining FEOM and the filtering framework, is used to study the parallel efficiency of the framework and of the filter algorithms. In addition, the filtering performance of the three error subspace Kalman filters is analyzed on the basis of twin experiments. These experiments extend the twin experiments performed in chapter 4 to a 3-dimensional test-case. The data assimilation experiments are performed with an idealized configuration of FEOM using a rectangular grid. Assimilated are synthetic observations of the sea surface height.

The major properties of the finite element model FEOM are described in section 9.2. Subsequently, in section 9.3, the configuration of the twin experiments is described in detail. The filtering performance of the three error subspace Kalman filters SEEK, EnKF and SEIK is examined in section 9.4. Here the abilities of the filter algorithms accurately estimate the 3-dimensional model fields is studied. The parallel efficiency of the framework and the filter algorithms is finally assessed in section 9.5.

9.2 The Finite Element Ocean Model FEOM

The finite element ocean model FEOM has been developed recently at the Alfred Wegener Institute [12]. It is a three-dimensional model designed to study the thermohaline circulation of the ocean on basin to global scales for periods from years to decades. The data assimilation framework introduced in chapter 8 permits to use this model as a 'black box' to perform the required model forecasts. In particular, the filter routines

are independent from the discretization method – finite elements, finite differences, or others – used to compute the forecasts.

A detailed description of FEOM has been given by Danilov et al. [12]. Here only the major properties of this model are summarized. FEOM is based on the primitive equations, see e.g. [72], which describe the thermo-hydrodynamics of the ocean. Namely, the primitive equations govern the velocity field $(\vec{u}, w) = (u, v, w)$, the sea surface height ζ, and the baroclinic pressure anomaly p. Further, the sea water density $\rho_0 + \rho$, where ρ_0 is the mean density, the temperature field T, and the salinity field S are described in the spherical coordinate system (λ, θ, z) by the equations

$$\partial_t \vec{u} + f(\vec{k} \times \vec{u}) + g\nabla\zeta - \nabla \cdot A_l \nabla \vec{u} - \partial_z A_v \partial_z \vec{u} = -\frac{1}{\rho_0}\nabla p + (\vec{u}\nabla + w\partial_z)\vec{u} \ , \tag{9.1}$$

$$\partial_z w = -\nabla \cdot \vec{u} \ , \tag{9.2}$$

$$\partial_t \zeta + \nabla \cdot \int_{z=-H_0}^{z=\zeta} \vec{u} dz = 0 \ , \tag{9.3}$$

$$\partial_z p = -g\rho \ , \tag{9.4}$$

$$\partial_t T + \nabla \cdot (\vec{u}\ T) - \nabla \cdot \kappa_l^T \nabla T - \partial_z \kappa_v^T \partial_z T = 0 \ , \tag{9.5}$$

$$\partial_t S + \nabla \cdot (\vec{u}\ S) - \nabla \cdot \kappa_l^S \nabla S - \partial_z \kappa_v^S \partial_z S = 0 \ , \tag{9.6}$$

$$\rho - \varrho(T, S, p) = 0 \ . \tag{9.7}$$

Here, f is the Coriolis parameter and $\vec{k}$ is the vertical unit vector. A_l, A_v are the lateral and vertical momentum diffusion coefficients. g is the gravitational acceleration. κ_l^T and κ_v^T are the lateral and vertical diffusion coefficients for the temperature. The corresponding coefficients for the salinity are κ_l^S and κ_v^S. The bottom of the ocean is at $-H_0(\lambda, \theta)$. $\varrho(T, S, p)$ denotes the equation of state. It is used to compute the density ρ from the temperature, salinity, and pressure fields.

The primitive equations are discretized on an unstructured grid with variable resolution. This 3-dimensional grid is built by tetrahedral elements. It is generated from a 2-dimensional triangular grid at the ocean surface which defines vertical prisms. Elementary prisms are obtained by subdividing the vertical prisms by level surfaces. The elementary prisms are split into tetrahedrons. The model fields are approximated using linear basis functions on these elements. A backward Euler method is used for the time stepping. The system of linear equations, which results from the finite element discretization, is solved by algorithms which are implemented in FEOM using the Family of Simplified Solver Interfaces (FoSSI) by Frickenhaus et al. [23]. FoSSI provides common interfaces to various solver libraries for sparse systems of linear equations like PETSc [64] or the solver PILUT by Karypis and Kumar [43].

Danilov et al. [12] tested the model performance in a configuration for the North Atlantic. Due to the size of 86701 nodes of the 3-dimensional grid, it is not feasible to use this configuration for the data assimilation and speedup experiments performed here. For this reason, the experiments employ an idealized configuration of FEOM. The configuration uses linear density stratification and a linear equation of state $\varrho(T, S, p)$. Further, convection is neglected and the rigid-lid approximation is used. The model

domain is given by a rectangular box geometry with a structured grid. It is shown in figure 9.1. The box is centered at 44.5° north and occupies an area of 9 by 9 degrees. It has a depth of 4000m. The discretization comprises 11 vertical levels and a horizontal grid of 31 by 31 points. This amounts to 10571 nodes of the 3-dimensional grid and 961 surface nodes. The time evolution is performed with a time step of 3 hours. The salinity field is chosen to be constant over the model domain. The state vector for the filters consists of the zonal and meridional velocity components $\mathbf{u}$, $\mathbf{v}$, the temperature $\mathbf{T}$, and the sea surface height $\boldsymbol{\zeta}$. Apart from the 2-dimensional sea surface height, all of these are 3-dimensional fields. Hence, the state dimension amounts to $n = 32674$.

9.3 Experimental Configurations

To extend the examination of filtering performance presented in chapter 4 and to study the parallel efficiency of the filter algorithms, identical twin experiments are performed with the idealized configuration of FEOM. Synthetic observations only of the sea surface height are assimilated. The physical process which is simulated in the assimilation experiments is the propagation of interacting baroclinic Rossby waves. The waves are initialized with two horizontally localized columnar temperature anomalies of the same amplitude but opposite sign. This initialization is shown in figure 9.2. Propagating westward, the anomalies become deformed. They tilt toward each other via the induced velocity field. That is, a negative temperature anomaly produces a counterclockwise rotation in the upper levels and a clockwise rotation in the lower levels. The rotation of a positive temperature anomaly is vice versa. These opposing rotations introduce non-linearity which is necessary to test the filtering performance of the error subspace Kalman filters.

The data assimilation experiments are conducted over a period of 40 days. The interval between subsequent analyses is set to 2.5 days. For the twin experiments the "true" state trajectory is generated by integrating the initialization displayed in figure 9.2 over a period of 45 days. To generate synthetic observations of the sea surface height, Gaussian noise with constant variance of 0.01 m^2 is added at each time step to the sea surface height field of the true state sequence. The amplitude of the temperature anomalies, and thus of the sea surface height, decreases over time. This is caused by diffusion. Hence, the relative noise amplitude of the observations increases during the assimilation period. Initially the standard deviation of the noise in the observations is at about 20 percent of the amplitude of the true surface height. After 45 days, the errors in the observations increased to about the same level of the surface height itself. The generated observations are used with an offset of 5 days in model time. Assimilating only observation of the sea surface height, the dimension of the observation vector amounts to $m = 961$. Figure 9.3 compares the observed sea surface height field with the true one at the initial time of the experiments. The observation errors are clearly visible, but also the observed information is apparent.

To initialize the filter in the twin experiments, the covariance matrix of 2268 state vectors is computed. These vectors are generated by 28 model forecasts using different initial locations of the temperature anomalies. Further, an additional variance of the

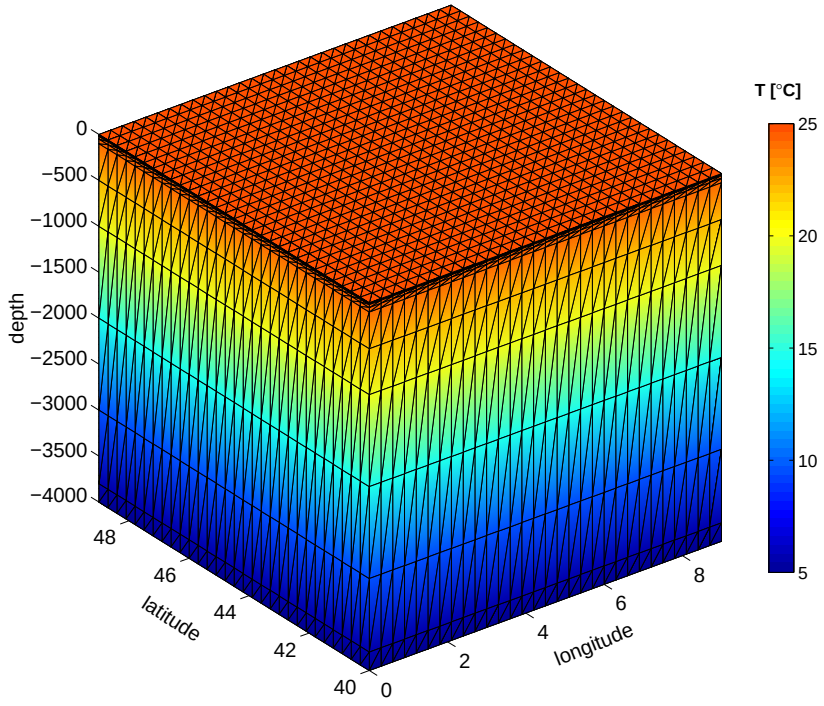


Figure 9.1: FEOM model grid used for the data assimilation experiments. It consists of 10571 nodes. Vertical levels are at the surface and in the following depths: 7.5, 20, 50, 100, 500, 1000, 2000, 3000, 3800, and 4000 meters. The coloring shows the linear temperature stratification.

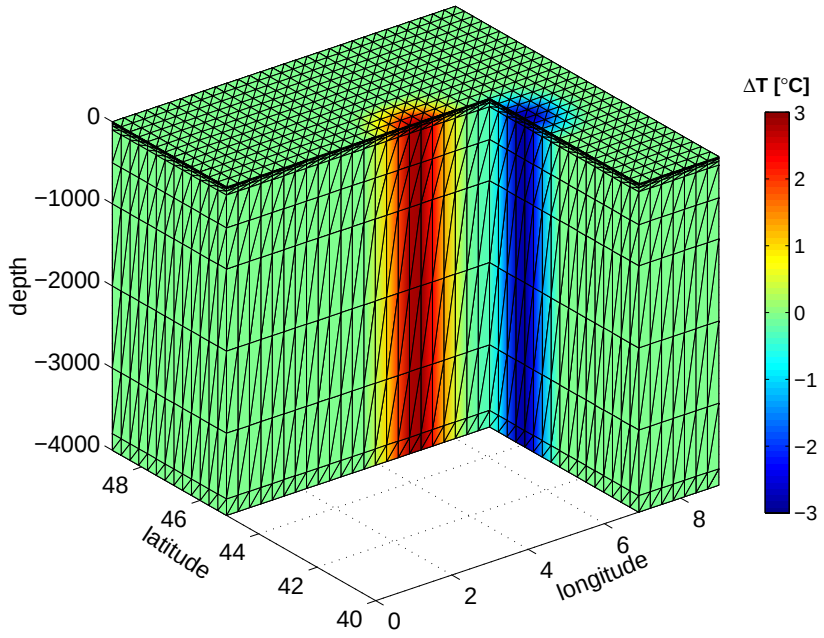


Figure 9.2: Cut into the model grid showing the temperature anomalies.

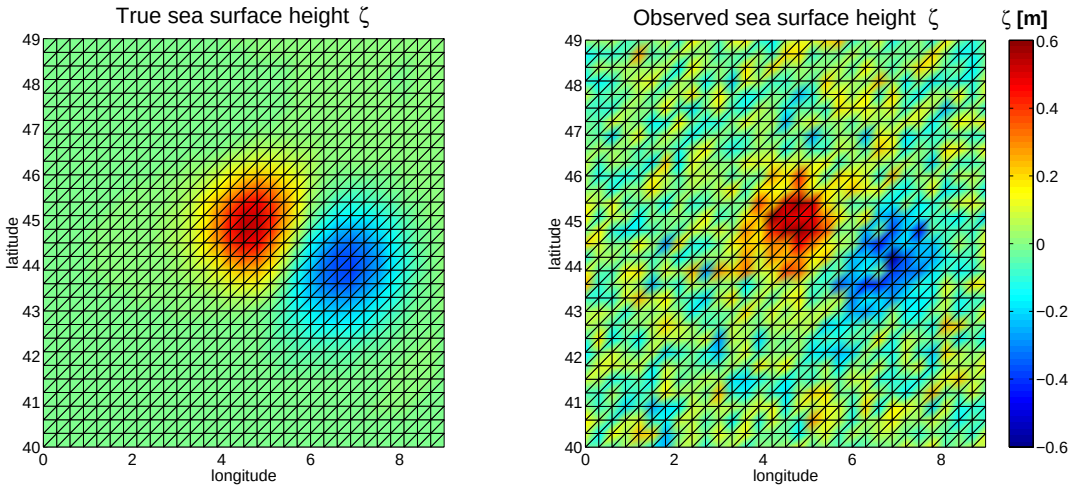


Figure 9.3: Comparison of the true (left) and the observed (right) sea surface height field $\boldsymbol{\zeta}$ at the initial analysis update.

sea surface height fields of 0.1m is assumed. The obtained covariance matrix, which describes the temporal variations and correlations of the model fields, is used as the initial error estimate in the filter experiments. The initial state estimate for the twin experiments is chosen as the mean state of the 28 model runs. The generation of the state ensembles for SEIK and EnKF and the initialization of the mode matrix for SEEK is performed as described for the experiments with the shallow-water-equation model in chapter 4. To examine the abilities of the filter algorithms to estimate the true state from the chosen initial state, an evolution of the initial state estimate is performed without assimilating observations. This state sequence is denoted the "free" state trajectory.

To simulate model errors in the application of the EnKF and SEIK filters, a wind forcing field of two gyres is applied whose shape and amplitude are controlled by two parameters. To obtain a stochastic forcing, these parameters are initialized by random numbers. Each ensemble member was forced by a different wind field which was constant over the forecast period. To retain comparability, the SEEK filter was used without a forgetting factor, since this could be applied to all three filters, or explicit treatment of a model error covariance matrix. Thus, the twin experiments using SEEK are performed without consideration of model errors.

Most of the computation time is spent in evolving the model states. Since the computation time is usually a limiting factor in data assimilation problems, results for assimilation experiments are compared in which all filters perform the same number of model evaluations. This configuration provides also comparable execution times for assessing the parallel efficiency of the three filter algorithms. To obtain configurations with equal numbers of model evaluations, the rank r used in SEEK and SEIK is set to $r = N - 1$ where N is the ensemble size of the EnKF.

The experiments have been performed on a Sun Fire 6800 server with 24 processors, each having a frequency of 1050 MHz. The experiments in section 9.4 used the solver PILUT while the experiments in section 9.5 used PETSc. This different choice was motivated by the fact that the use of PILUT resulted in inferior speedup values than PETSc. In contrast to this, the assimilation experiments with the PILUT solver provided a better filtering performance than those using PETSc. Since this work is not aimed at the optimization of the model, the solver was chosen depending on the best results either in terms of filtering performance or in terms of speedup.

9.4 Filtering Performance

Before the parallel efficiency of the filter algorithms is studied in section 9.5, the filtering performance of the SEEK, SEIK, and EnKF algorithms is assessed for their application to the configuration of FEOM described in the preceding sections. These experiments extend the 2-dimensional experiments of chapter 4 to a 3-dimensional test-case.

9.4.1 Reduction of Estimation Errors

For an ensemble size of $N = 60$, figure 9.4 shows the rms deviation E_1 of the assimilated state from the true state normalized by the rms deviation of the free state from the true state. The deviation is computed over all grid nodes with equal weights for all nodes. Thus, no volume-normalization is performed which would consider the different distances between neighboring levels of the model. The relative estimation error is displayed separately for the four state fields. For $N = 60$, the EnKF and SEIK filters yield comparable results. For smaller ensembles, the difference of E_1 for the two filters is larger, with the EnKF performing worse than the SEIK filter (not shown). This can be expected because of the inferior sampling quality of the Monte Carlo sampling applied to initialize the EnKF algorithm. Since the difference of the sampling quality decreases for larger ensembles, the results of EnKF and SEIK become almost identical for larger ensembles. The SEEK filter shows a behavior distinct from the two other algorithms. This behavior is caused by the forecast scheme of the SEEK filter which applies a gradient approximation of the linearized forecast of the covariance modes. For all model fields the relative estimation errors tend to increase toward the end of the assimilation period. This is due to the growing relative error level in the observations which is discussed in section 9.3.

The largest error reduction is obtained for the sea surface height ζ. As observations of the sea surface height are assimilated, this field is expected to show the smallest normalized estimation error of the four model fields. To get an idea of what represents the achieved reduction of the relative estimation error to about 0.27 for the sea surface height, the left hand side of figure 9.6 shows in the uppermost panel the true sea surface height at the end of the assimilation period. In the middle panel, ζ is shown as estimated by the EnKF filter with $N = 60$. The sea surface height which is obtained from the free evolution, i.e. when the initial state estimate is evolved without assimilation, is displayed in the lowermost panel. The sea surface height estimated by the

EnKF algorithm reproduces accurately the shape of the true $\boldsymbol{\zeta}$. The locations of the minimum and the maximum are well estimated. The amplitudes are underestimated by about 10%. In contrast to this, the sea surface height without assimilation deviates strongly from both the true and SEIK-estimated $\boldsymbol{\zeta}$.

The velocity components $\mathbf{u}$ and $\mathbf{v}$ are updated via the estimated cross correlations between the sea surface height and the velocity components. Despite this, the relative estimation errors of the meridional velocity component $\mathbf{u}$ are of comparable size to

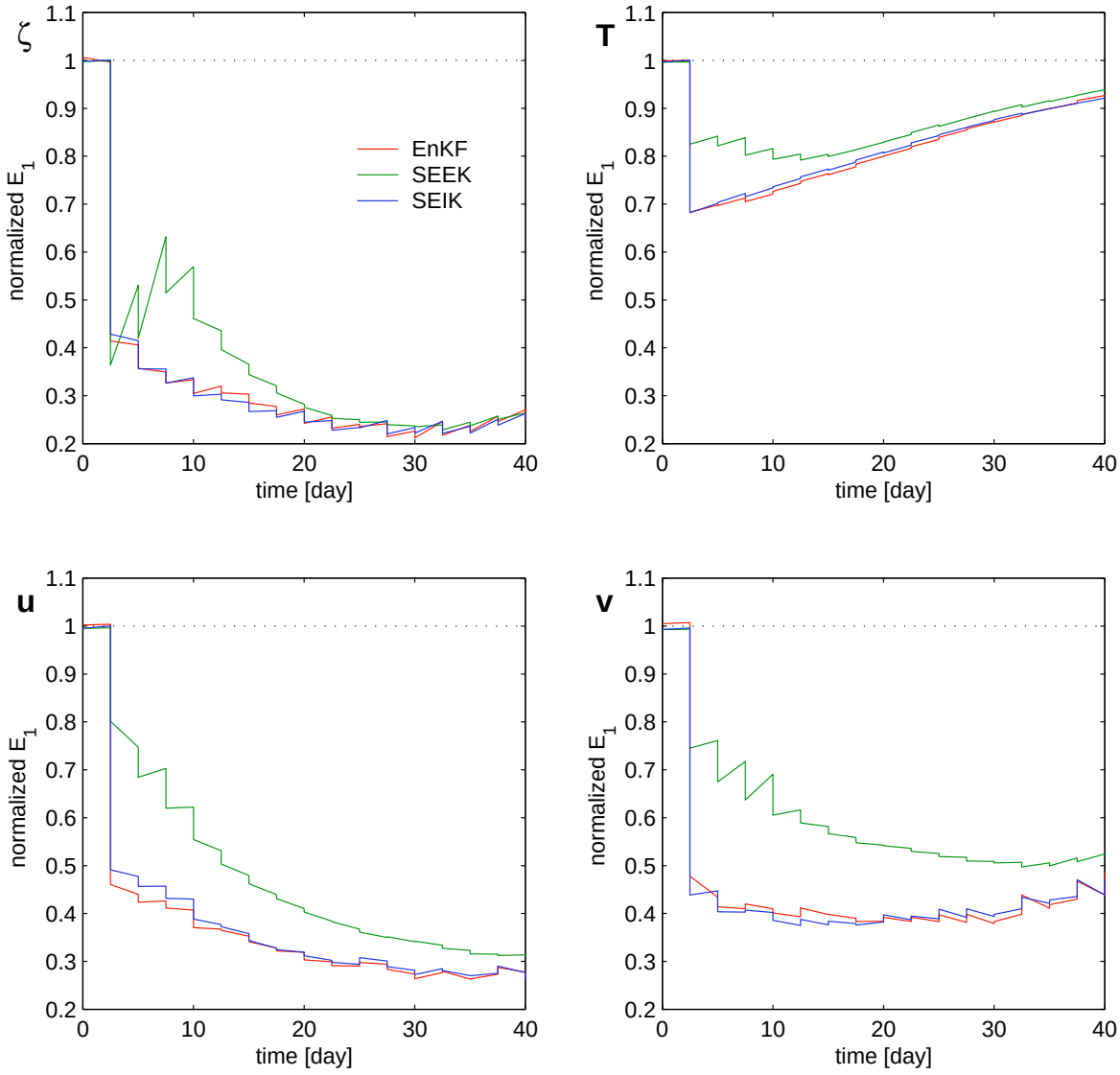


Figure 9.4: Time dependence of the relative estimate errors E_1 for experiments with $N = 60$. Shown is E_1 separately for the sea surface height $\boldsymbol{\zeta}$ (top left), the temperature field $\mathbf{T}$ (top right), and the two components $\mathbf{u}$, $\mathbf{v}$ (respectively on the left and right hand sides of the bottom row) of the velocity fields.

those of the sea surface height in the case of EnKF and SEIK. This relation shows, that the cross covariances are estimated quite well by the nonlinear ensemble forecast. In contrast to this, the linearized forecast performed in SEEK yields much worse estimates of the cross covariances. This can be deduced from the much larger estimation errors for **u** obtained with SEEK.

The estimate of the zonal velocity component **v** is less precise than the estimate of **u** for all three filters. After the first analysis phase, the estimation error of both velocity components is of comparable size. While the estimation error for **u** decreases during the course of the assimilation experiment, the estimation error for **v** remains at a level of about 0.4 when using the EnKF or the SEIK filter. Thus, the cross covariances are not estimated sufficiently precise to further decrease the error level for this velocity component. During some analysis updates, e.g. at day 25, the estimation error increases. In this case the estimated cross covariances have the wrong sign.

The relative estimation error of the temperature field **T** shows a behavior distinct from the other model fields. The error reduction at the first analysis update is smaller for **T** than for the other fields. For the EnKF and SEIK filters, the relative estimation error of the temperature field increases immediately after the first analysis update. Further, the estimation error remains almost unchanged during the analysis update. Thus, no useful estimates of the cross correlations are available after the first analysis update. The estimates of variances and correlations within some model field are typically much more precise than estimated cross correlations. Thus, even a limited number of temperature measurements would enhance the estimation quality of the temperature field for all three filters.

9.4.2 Estimation of 3-dimensional Fields

To examine the ability of the filter algorithms to estimate the 3-dimensional model fields by assimilating only surface measurements profiles of the relative estimation errors at the end of the assimilation period are shown in figure 9.5. The values displayed in the diagrams are the normalized rms estimation errors computed over single levels of the model.

The profiles for the two velocity components **u** and **v**, displayed on the left and middle panels, show a small relative estimation error from the surface to -1000m depth. Below -3000m the estimation error is a also small, but it increases toward the bottom. At the depth of -2000m the estimation error shows a maximum. For the experiments with SEIK and EnKF, this maximum is even larger than one. The estimation errors obtained with SEEK are of similar size to those achieved by the EnKF and SEIK filters. They are, however, larger at all depths, except at -2000m. For all three filters, the relative estimation errors are smaller for the meridional velocity component **u** than for the zonal velocity **v**.

The peak in the relative estimation error at the depth of -2000m is due to the normalization by the estimation error of the evolution without assimilation. As has been described in section 9.3, the temperature anomalies generate a counterclockwise rotation in the upper levels and a clockwise rotation in the lower levels. The turning

point of these rotations is approximately at the depth of -2000m. Due to this, the velocities are minimal at this depth in the true state, the free state and the assimilated states. This causes minimal rms deviations of the velocities of the free evolution from the velocities of the true evolution. Without normalization, the estimation errors of the assimilated velocities are of comparable size to those of the non-assimilated velocities at -2000m depth. Due to the normalization, the estimation errors appear larger than their absolute value.

The increase of the relative estimation error below -3000m is not due to the normalization, as the absolute estimation errors also increase below -2000m depth. Thus, the quality of covariances between the sea surface height and the velocity fields is worse in the deep ocean than for the upper levels. Overall, all three filters show good abilities to reduce the estimation error of the velocity field also in the lower levels of the model. The level -2000m appears to be a rather pathological situation which the algorithms cannot handle well.

The profile of the relative estimation errors of the temperature field, shown on the right hand side of figure 9.5, exhibits a different dependence on depth than the estimation errors of the velocity field. In the uppermost levels the estimation error of

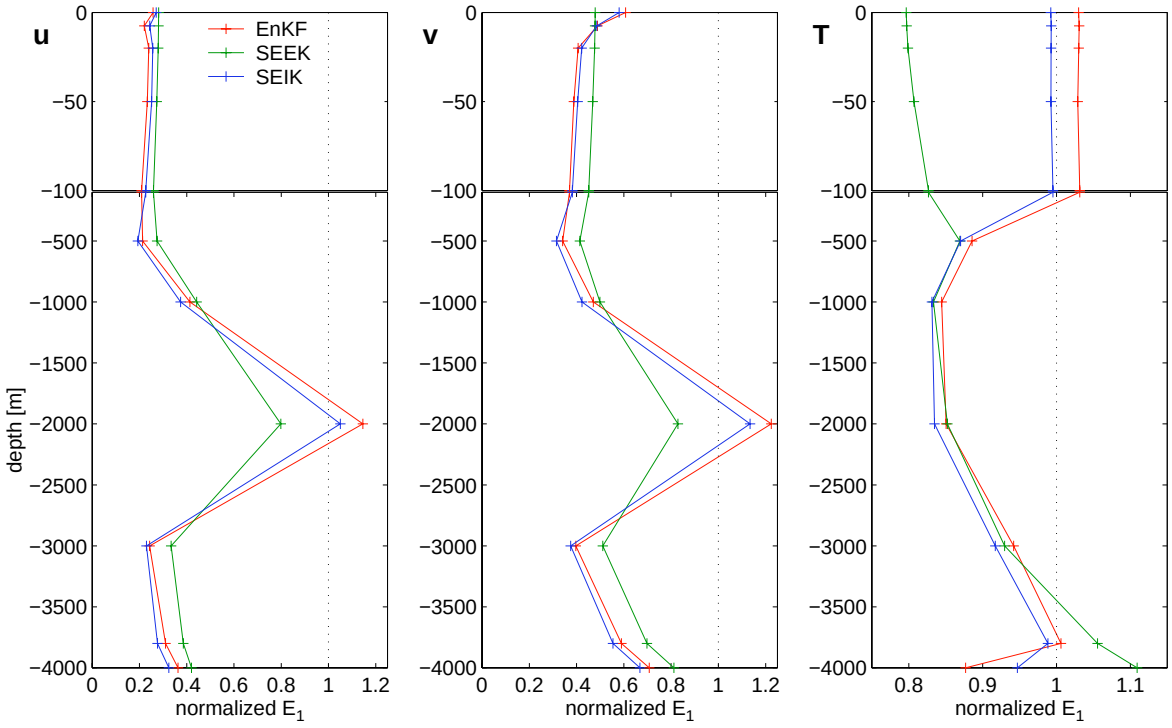


Figure 9.5: Profiles of the rms estimation errors of single layers normalized by the corresponding rms deviation of the free state from the true state for $N = 60$. Shown are the two components **u**, **v** of the velocity fields and the values for the temperature field **T** at the end of the assimilation period.

the temperature field is not reduced by the SEIK and EnKF algorithms. In contrast to this, the relative estimation error is decreased to a level of about 0.8 when the SEEK filter is applied. Between -100m and -2000m all three filters reduce the estimation error to similar level of about 0.85. Below -2000m the relative estimation error increases for all three filter algorithms to a level around unity.

The large relative estimation errors in the uppermost 100 meters are misleading. This becomes apparent from the panels on the right hand side of figure 9.6. The uppermost panel shows the true temperature field at a depth of -50m. The panel in the middle shows the temperature field as estimated by the EnKF with $N = 60$. For comparison, the free temperature field is displayed in the lowermost panel. The shape of the estimate from the EnKF reproduces the shape of the true temperature field rather well. The amplitude of the positive temperature spot is, however, over-estimated. The free temperature field is distinct by showing only a single positive temperature anomaly.

In the level at -500m and below the temperatures are generally over-estimated by about 0.1°C. This is displayed in figure 9.7 which shows the temperature fields analogous to the right hand side of figure 9.6 for the levels at -1000m and -3800m. While the shape of the estimated temperature field is still reasonable at -1000m, this is no more the case for the level at -3800m. Here, the estimate resembles the shape of the free temperature field which is obtained from the evolution of the state estimate without assimilating observations. The assimilation has only a small influence on the temperature field at -3800m. Namely, the warm area with temperatures above 6.3°C is shifted further to the north-east. In addition, the temperature is decreased around (44°N, 7°E).

Overall, the three filter algorithms show a very limited ability to estimate the temperature correctly when only measurements of the sea surface height are assimilated. The shape of the temperature field is reproduced by the estimates in the upper 1000 meters. However, there is a bias in the temperature estimates. Due to this, additional temperature measurements, also in the depth, would be useful to obtain better estimates of the temperature field.

9.5 Parallel Efficiency of Filter Algorithms

Based on of the idealized configuration of FEOM, the parallel efficiency and the speedup of the parallel filtering framework is now examined. First, data assimilation experiments with a limited ensemble size are considered to assess the efficiency of the complete filtering framework. Subsequently, the parallel efficiency of the filter part is studied. For this experiments are conducted without time stepping. This reduces the computation time and hence permits to perform more experiments. In addition, the neglect of time stepping permits to examine also the efficiency of the domain-decomposed filter algorithms, while FEOM is not based on domain decomposition.

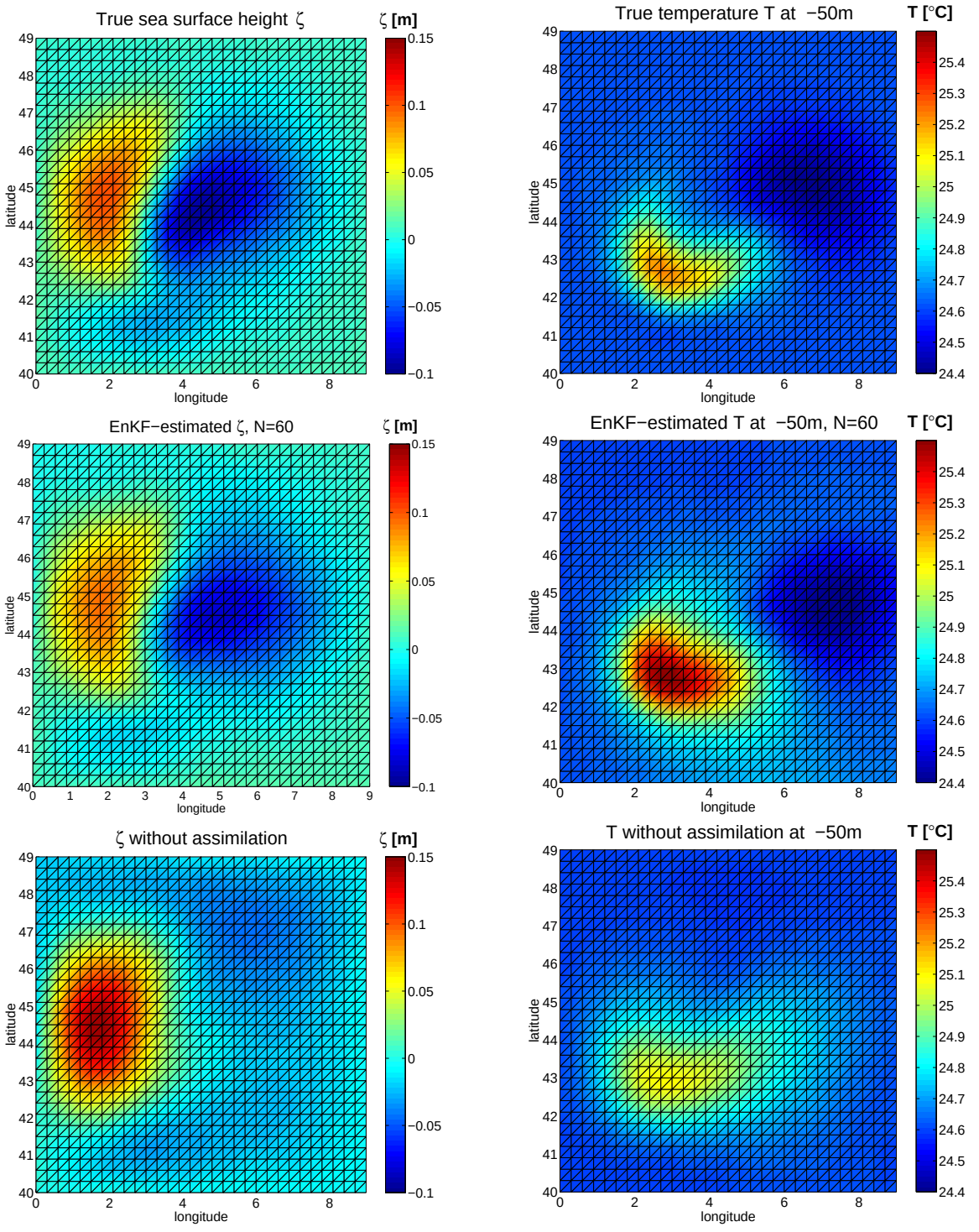


Figure 9.6: Comparison of true, estimated, and free model fields (from top to bottom) at the end of the assimilation period. The estimated field is shown for the EnKF with $N = 60$. The left hand side shows the sea surface height $\boldsymbol{\zeta}$. The temperature field $\mathbf{T}$ at a depth of -50m is shown on the right hand side.

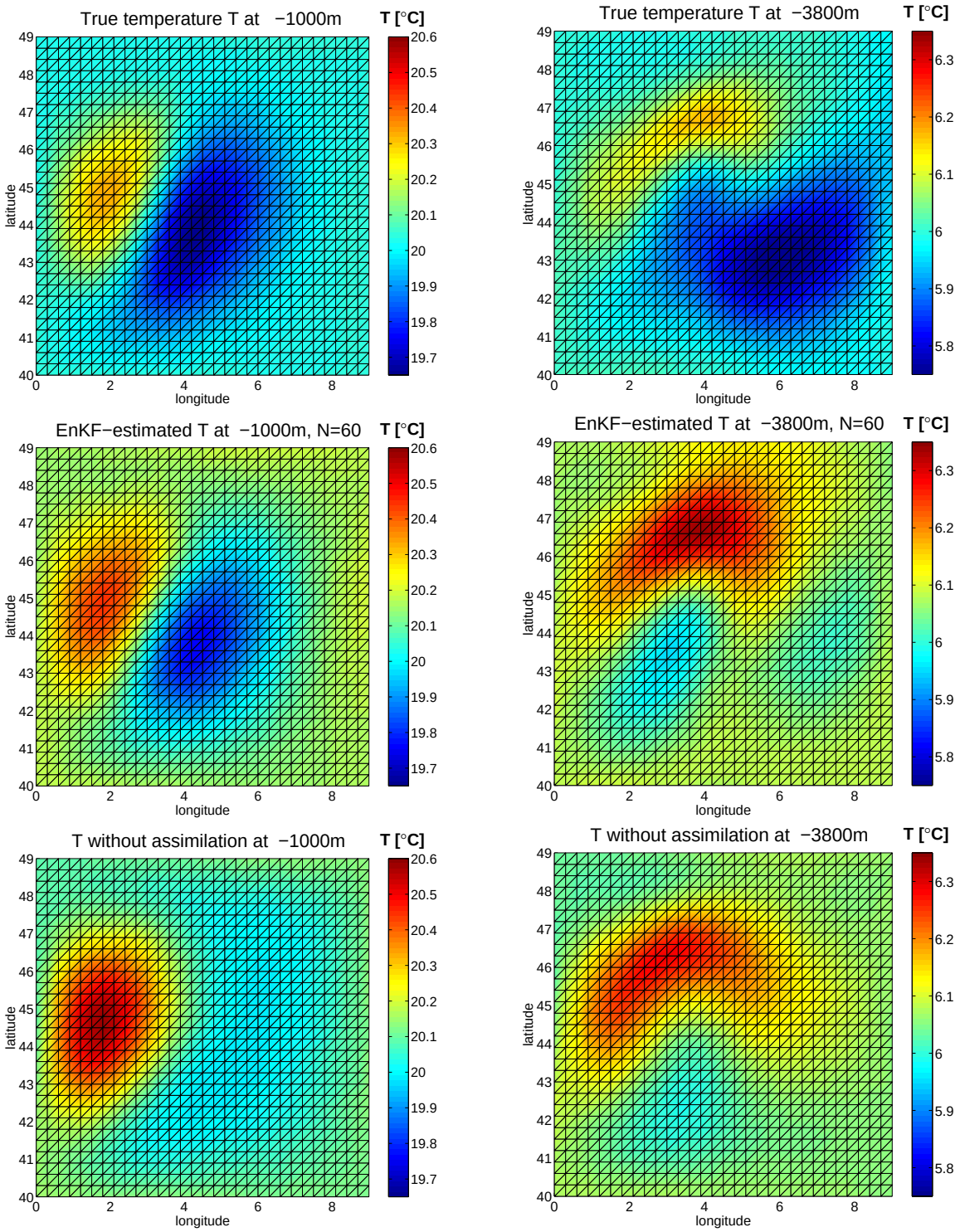


Figure 9.7: Comparison of true, EnKF-estimated, and free temperature fields (from top to bottom) at the end of the assimilation period. The right hand side shows the temperature field at a depth of -1000m; the left hand side just above the bottom at a depth of -3800m.

9.5.1 Efficiency of the Framework

To study the parallel efficiency and the speedup of the data assimilation framework, data assimilation experiments are performed with the three ESKF algorithms using different numbers of parallel model tasks. Since FEOM does not apply domain-decomposition, a configuration with mode-decomposed filters is applied. To reduce the computation time of the experiments in comparison to those in the preceding section, the data assimilation experiments are performed over a time period of 10 days. The interval between subsequent analyses is set to 12 hours. To compute the speedup, the state ensemble has to be divided evenly over the available model tasks. For this reason, an ensemble size of $N = 36$ ($r = 35$) is chosen. This ensemble size has the following properties:

- The ensemble is sufficiently large to provide a realistic data assimilation experiment. On the other hand, the ensemble is small enough to perform a large number of experiments.

- To assess the speedup, a large variety of different numbers of model tasks is required. To ensure that each model task evolves the same numbers of ensemble states, the chosen numbers of model tasks need to be divisors of the ensemble size. In addition, the number of possible parallel model tasks is limited due to a limited number of processors in the computer system used for the experiments. Using $N = 36$, the experiments can be executed with 1, 2, 3, 4, 6, 9, 12, 18, and 36 parallel model tasks. This enables efficient use of the available 24 processors of the Sun Fire 6800.

Using the configuration described above, the execution time for a single-processor, i.e. serial, experiment is about 9 hours on the Sun Fire 6800. The execution time decreased to about 35 minutes when 18 parallel model tasks are used. Using a single processor, the execution time for the EnKF algorithm was about 18 seconds. The analysis and the resampling phases of SEEK lasted respectively about 0.2 and 2.2 seconds. The analysis phase of SEIK took 0.4 seconds while the resampling phase lasted about 1 second. Thus, the analysis phase of SEIK is slower than that of SEEK, but the resampling phase is faster. This is consistent with the computational complexity of the algorithms which was discussed in section 3.4.

Figure 9.8 shows speedup and parallel efficiency for filtering experiments using the configuration of the framework where the filter is executed by one process of each model task. The speedup is computed from the total execution time of one series of experiments. Thus, the time for the initialization of the model and the filter are included as well as the time for the user analysis routines. The user analysis routines compute the filter-estimated variances and write the estimated state to a disk file. Each model task is executed by a single process. Hence, the total number of processes for an experiment equals the number of model tasks and the number of filter processes. This configuration has been chosen to allow for a maximal number of parallel model tasks. This choice does not limit the significance of the results when the speedup in relation

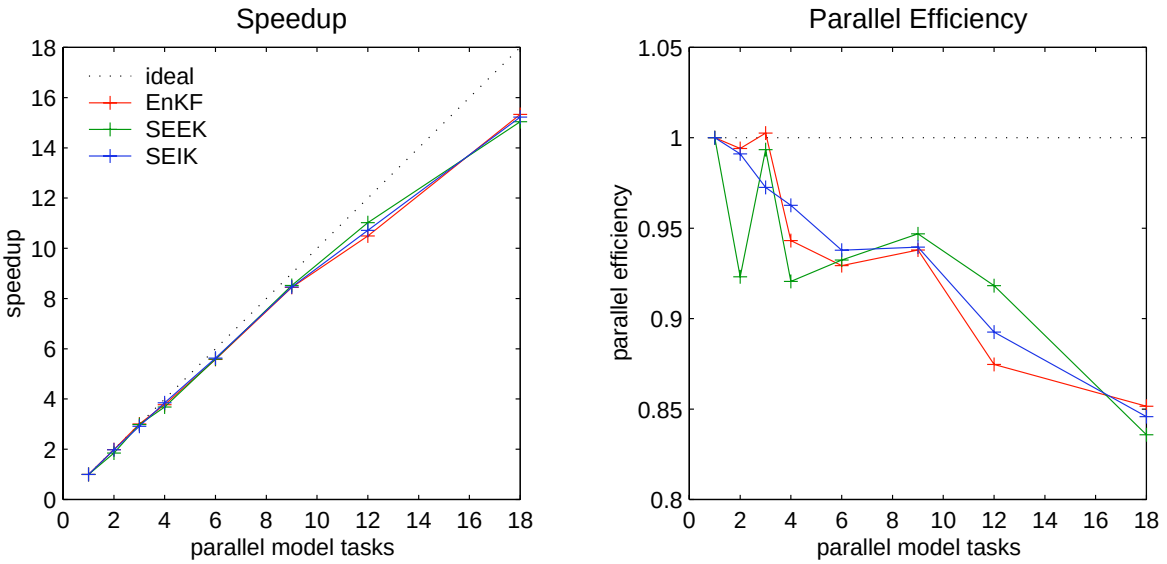


Figure 9.8: Speedup (left hand side) and parallel efficiency (right hand side) in dependence on the number of parallel model tasks for the framework with a filter process on each model task.

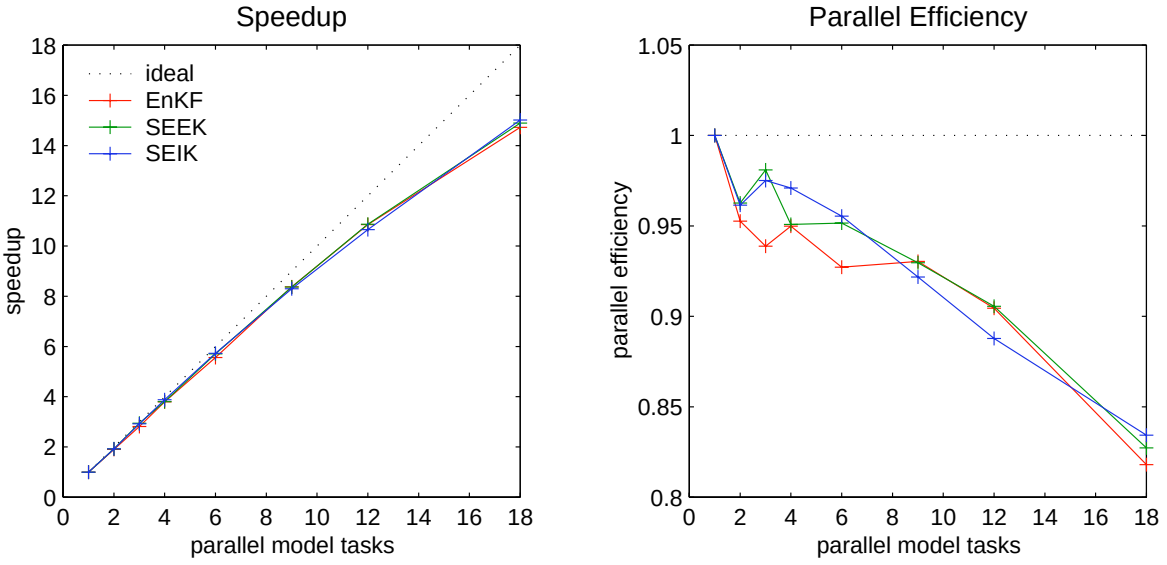


Figure 9.9: Speedup (left hand side) and parallel efficiency (right hand side) in dependence on the number of parallel model tasks for the framework with disjoint process sets for filter and model. The filter part is computed by a single process.

to the used number of model tasks is considered. Since here the number of processes in a model task does not change, the computation time for the forecast of a single state is independent of the number of parallel model tasks. Using a filter process on each model task minimizes the amount of communication between model and filter (see section 8.3.1). In fact, since each model task is executed by a single process, no communication between model and filter is conducted. Thus, the parallel efficiency of the program is limited only by the serial parts of the model and the filter algorithms, by the communication performed within the filters, and by possible different times to compute the forecast of different model states.

The speedup in figure 9.8 is excellent for all three filter algorithms. The small differences between the filters are not statistically significant. The sensitivity of the results was examined using 10-fold experiments with the same number of model tasks. Due to variations in the total execution time of the experiments, a standard deviation of about 3% results for the speedup. Thus, the filter framework yields equal values of the speedup for the three ESKF algorithms. The parallel efficiency of the data assimilation system decreases slightly when the number of parallel model tasks is increased. With 18 model tasks an efficiency of about 85% is obtained.

For comparison, figure 9.9 shows speedup and parallel efficiency for experiments using disjoint process sets for the model and filter parts of the program. In these experiments the filter is executed on a single process only. Thus, the parallel efficiency is limited by the serial operations of the filter, serial parts of the model, and by the communication required to exchange the state vectors between filter and model. Further, different computation times for the forecasts can limit the efficiency when other processes have to wait for one of the model tasks to complete its work.

Using disjoint process sets, the speedup is very similar to the speedup obtained by the configuration with a filter process on each model task. The small differences are again not statistically significant. The standard deviation of the speedup amounts again to about 3%. Due to these uncertainties no more detailed results can be drawn from the values of the speedup. In particular, it is not possible to determine which of the two process configurations, filter processes joint with the model processes or disjoint process sets for model and filter, is more efficient.

The deviation from an optimal parallel efficiency of the data assimilation system is caused by varying execution times of the state evolutions on different model tasks. Since the processes are synchronized at the end of a forecast phase, this desynchronization reduces the speedup of the forecast phase. The influence of the analysis and resampling phases are negligible. For the EnKF, which is the most costly of the three filter algorithms, the execution time for the analysis and resampling phases amounts to less than 0.1% of the total execution time for the serial experiment. In addition, the influence of the serial model initialization and the execution of the user analysis routine are negligible. These phases last respectively about 6 and 10 seconds in the serial experiment.

9.5.2 Speedup of the Filter Part for Mode-decomposition

Despite the fact that in the experiments conducted in the preceding section with the idealized configuration of FEOM the computation times for the filters were negligible, it is instructive to examine the speedup of the filter routines. It will be important when the computation time for the model is less dominant. This can occur, e.g., if observational data is frequently available causing the time interval between successive analysis phases to by very small.

To assess the parallel efficiency of the filter routines, data assimilation experiments without time stepping are performed. For this the call to the time stepper routine of FEOM is out-commented in the source code of the program used for the experiments in section 9.5.1. Apart from the time stepping, the experiments are analogous to the filtering experiments discussed in the preceding section. To obtain sufficiently large execution times of the filter routines, the analysis phase is performed 20 times. This corresponds to an interval of three hours between subsequent analyses in the experimental configuration with time stepping. To study the dependence of the parallel efficiency on the ensemble size, experiments with $N = 60$ and $N = 240$ are performed.

Figure 9.10 compares the execution time and the speedup for two different ensemble sizes for the update phase of the filters for mode-decomposed filter algorithms. The left hand side corresponds to an ensemble size of $N = 60$; the right hand side was computed with $N = 240$. For the SEEK and SEIK filters the timing includes the time for the analysis and the resampling phases. The serial experiments have also been performed with the parallel filter routine. Thus, the used routines were not optimized for serial computations. The MPI operations were called also in the serial experiments. The execution time for these operations is much shorter in this case, but still there is a small overhead due to these redundant operations.

For $N = 60$, the SEEK and SEIK filters are much faster than the EnKF algorithm. The fastest algorithm is the SEIK filter. This is due to the much faster resampling phase of SEIK compared with SEEK. In the serial experiments, the analysis phase of SEEK takes about 0.6 seconds while the resampling lasts about 10.5 seconds. The analysis phase of SEIK is longer than that of SEEK taking 0.9 seconds. However, the resampling phase of SEIK lasts only 4.3 seconds. In these experiments, the resampling phase of the SEEK filter is executed after each analysis. As was discussed in section 2.4.1, this is actually not necessary. Thus, performing the resampling in SEEK less frequently could significantly speed up this algorithm. The small speedup of EnKF is partly due to the generation of the observation ensemble. Since only a single observation vector is read from a file, the observation ensemble has to be generated by the transformation of independent random numbers which was discussed in section 4.2. The generation of the observation ensemble took about 26 seconds for $N = 60$. The algorithm itself lasted about 17 seconds. But, even if the time required for the initialization of the observation is neglected, the EnKF algorithm would remain the slowest algorithm. This is caused by the solver step for the representer amplitudes (line 20 in algorithm 7.3). The complexity of this operation scales with $\mathcal{O}(m^3 + m^2 N)$ as was discussed in section 3.4. Other influences on the speedup of the EnKF algorithm will be discussed below.

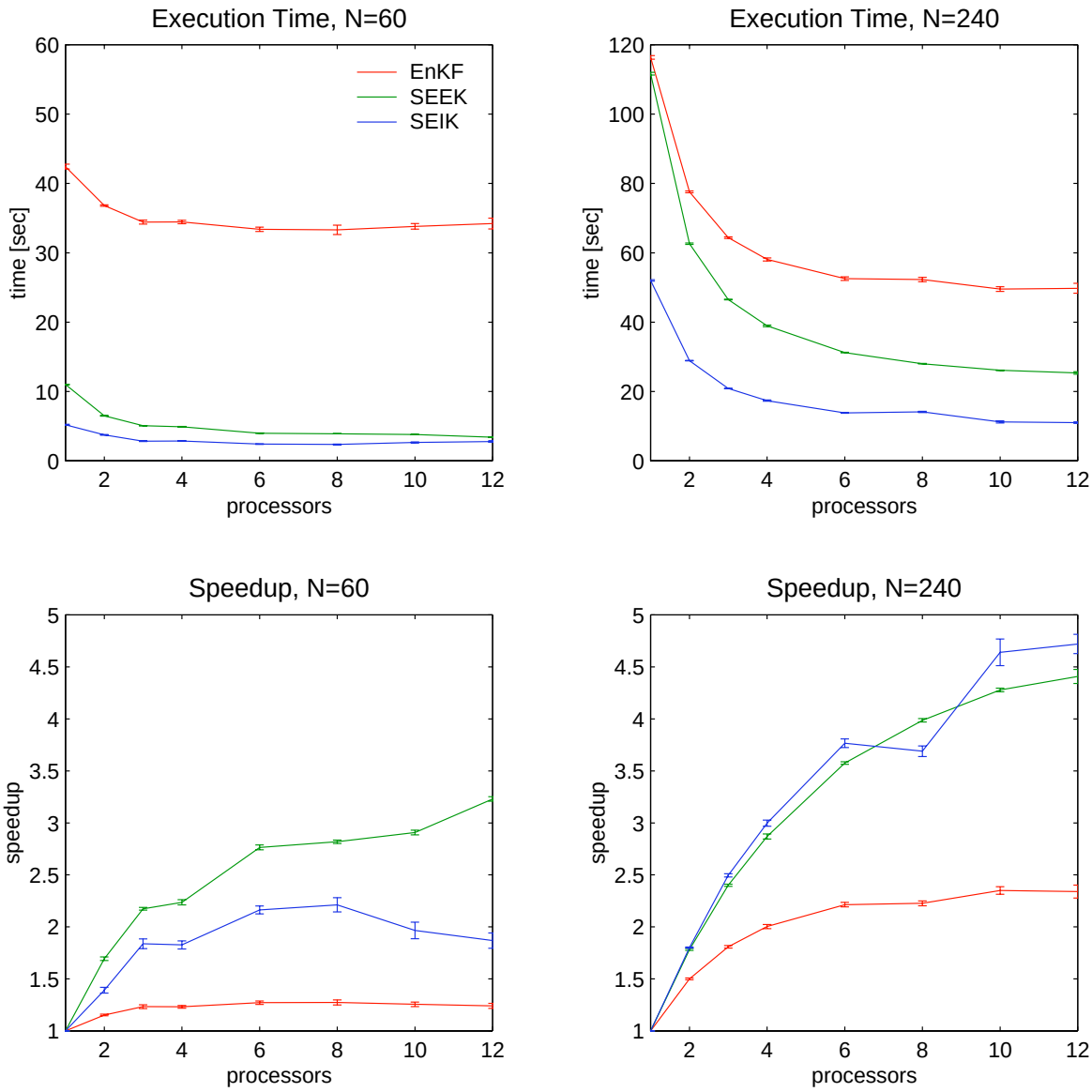


Figure 9.10: Execution time and speedup for the filter update phases in dependence on the number of processes. In the experiments, the mode-decomposed filter was applied. Displayed are mean values and standard deviations over ten experiments for each combination of filter algorithm and number of processes. The left hand side shows results for $N = 60$, the right hand side for $N = 240$.

The relative differences in the execution times are smaller for $N = 240$ than for $N = 60$. Using the larger ensemble size, the SEIK filter remains the fasted algorithm while the EnKF algorithm is still the slowest filter, even if the generation of the observation ensemble is neglected. The execution time for the EnKF triples while that for SEEK and SEIK increases tenfold. The small increase in the execution time

for the EnKF is due to the fact that the time for the initialization of the observation ensemble only approximately doubles since here several operations do not dependent on the size of N. The time for the remaining part of the EnKF quadruples. The increase in the execution time of SEIK is dominated by the computation of the new ensemble matrix in line 10 of the resampling algorithm 7.5. For SEEK, the increase in time is also dominated by the resampling phase. Here most of the time is spent in the computation of $\mathbf{T1_p}$ in line 8 of algorithm 7.2 and the computation of the new modes in line 15.

The speedup of the mode-parallel filter algorithms is rather disappointing. This becomes apparent from the bottom row of figure 9.10 which shows the speedup for the experiments with $N = 60$ and $N = 240$. The fluctuations in the speedup are mainly due to cache-effects of the computer used for the experiments. Therefore, the numerical efficiency of matrix-operations like matrix-matrix products depends on the dimensions of the involved matrices. For $N = 60$, the best speedup is obtained with the SEEK filter. Using 12 processes, a speedup of about 3.2 is obtained which corresponds to a parallel efficiency of 27%. The worst speedup is exhibited by the EnKF algorithm. It stagnates at a value of about 1.2 when 12 processes are used. This corresponds to a parallel efficiency of 10%. The speedup is slightly better for the large ensemble size of $N = 240$. Here the speedup for SEEK and SEIK reaches respectively 4.4 and 4.7. Thus an efficiency between 37% and 39% is obtained with 12 processes. The speedup of EnKF is twice as large as for $N = 60$ stagnating at a value of about 2.4 with 12 processes.

The low parallel efficiency of SEEK and SEIK is mainly due to the extended communication which is needed in the algorithms. For increasing ensemble size, the time for computations increases relative to the time for communications. Thus the parallel efficiency increases for larger ensembles. The distinct efficiency of SEEK and SEIK for $N = 60$ is due to the different number of operations performed in their resampling phases. The amount of communication in the resampling phases of both algorithms is practically equal for $N = 60$. Since SEIK performs less operations, the allgather operation for $\mathbf{X}$ in line 6 of algorithm 7.5 is more dominant for the execution time than the allgather operation performed for $\mathbf{V}$ in SEEK. Since the time to perform the allgather operation increases with an increasing number of processes, the efficiency decreases for a larger number of processes. Using more than 6 processes, the allgather operation in SEIK lasts even longer than the computation of the new ensemble states. Therefore, the execution time of SEIK increases if the number of processes exceeds a value of 8. Hence, the speedup of SEIK decreases for the experiments using more than 8 processes.

For models with larger state dimension n, the speedup of the SEEK and SEIK filters will also be limited by the required initialization of the full ensemble or mode matrix by allgather operations. Also the differences between SEEK and SEIK will remain for increasing n, since the amount of communication and the complexity of the most expensive floating point operations in the resampling algorithm scale both with $\mathcal{O}(n)$.

The minor speedup of the EnKF filter is due to several factors. To examine the reasons in detail, the execution time and the speedup of different groups of operations are displayed in figure 9.11 for the EnKF with $N = 240$. In the serial experiment,

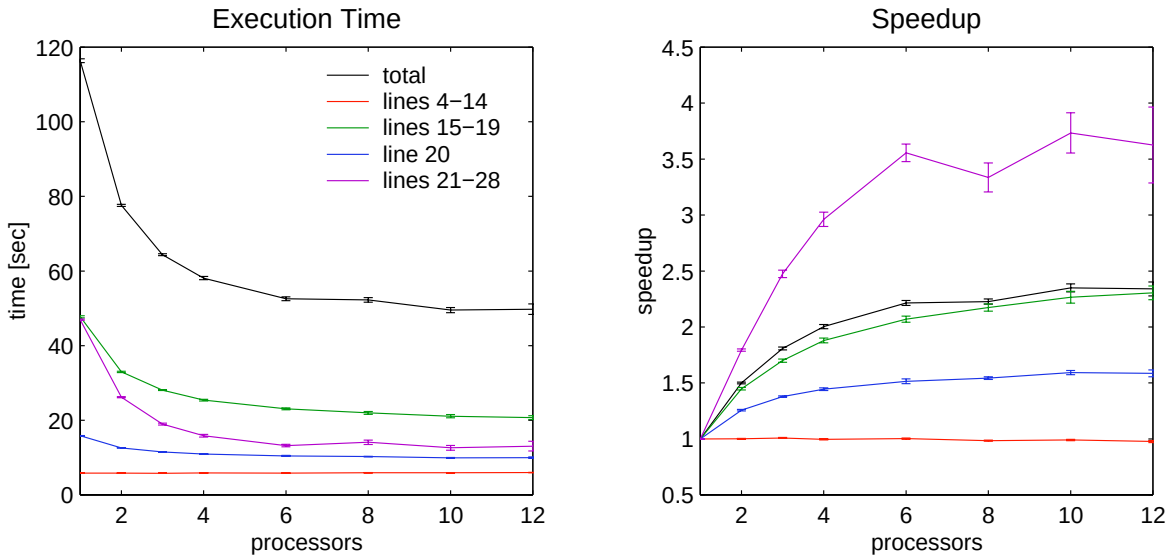


Figure 9.11: Execution times and speedup for the groups of operations in the EnKF analysis algorithm for $N = 240$. Shown are means and standard deviations analogous to figure 9.10. The line numbers given in the legend of the diagrams refer to those in algorithm 7.3.

the generation of the observation ensemble and the initialization of the residual matrix (lines 15 to 19 in algorithm 7.3) take together about the same time as the ensemble update with its preparations (lines 21 to 28). The the ensemble update shows a better speedup than the initialization of the residuals. The speedup for the ensemble update does, however, stagnates at a value of about 3.5. This is due to the allgather operation performed to initialize the matrix $\mathbf{T5} \in \mathbb{R}^{n \times N}$. The generation of the observation ensemble does also show a limited speedup since this operation requires the eigenvalue decomposition of the observation error covariance matrix $\mathbf{R} \in \mathbb{R}^{m \times m}$. The decomposition is independent of the local ensemble size and is not parallelized. The speedup of the other parts of the EnKF algorithm is worse than the ensemble update and the initialization of the residual matrix. The computation of matrix $\mathbf{T3} \in \mathbb{R}^{m \times m}$ in line 13 takes about 97% of the execution time of the operations in lines 4 to 14. Since this operation is not parallelized, the speedup for this part of the algorithm will be approximately constant with a value of one. The complexity of the solver step for the representer amplitudes in line 20 is $\mathcal{O}(m^3 + m^2 N)$. It is dominated by the LU-decomposition of the matrix $\mathbf{T3}$ which is performed by the LAPACK routine *DGESV*. Thus, the achievable speedup of the solver step is very small.

Overall, this discussion showed that the small speedup for the EnKF is caused by a combination of high amounts of communication and operations which are performed serially or do not have a good scalability in terms of performance. The speedup of the ensemble update could be major if the communication was faster relative to the computations. The solver step in line 20 and the computation of $\mathbf{T3}$ in line 13 will, however, remain a limiting factor for the parallel efficiency of the EnKF algorithm. The

speedup will be major if the dimension of the observation vector relative to the state dimension is smaller. This can be achieved by using a EnKF analysis algorithm which sequentially assimilates batches of observations as has been discussed in section 3.4. In addition, a better speedup can be expected for larger models if the amount of observational data remains constant.

9.5.3 Speedup of the Filter Part for Domain-decomposition

The experiments of the preceding section have been repeated using the domain-decomposed filter algorithms developed in section 7.3. Figure 9.12 shows execution time and speedup for the update phase of the filters. As in figure 9.10, results for $N = 60$ are displayed on the left hand side and results for $N = 240$ are shown on the right hand side.

The execution times for domain-decomposed filters look rather similar to those for the mode-decomposed filters. For the serial experiments, the times are about the same size. There are small differences due to the different number of communication operations which are even called if the filters are executed by a single process. A relevant difference to the experiments with mode-decomposed filters is the stronger decrease of the execution times with an increasing number of processes which is visible for SEEK and SEIK.

This behavior is quantified by the speedup. For $N = 60$ the SEEK and SEIK filters show an ideal, even slightly super-linear speedup. The super-linear speedup is caused by some operations which exhibit super-linear speedup. An example is the computation of the matrix $\mathbf{T1_p}$ in the SEEK resampling algorithm 7.7. This operation reaches a speedup of 14.8 with 12 processes. The super-linear speedup is caused by the effect that the local part of a decomposed matrix might fit better into the processor caches of the computer than the full matrix. Thus, the caches can be used more efficiently if the matrix is decomposed. In this case, the parallel efficiency of the operation will by larger than one. Whether a super-linear speedup occurs is dependent on the cache sizes of the computer system used for the experiments.

For $N = 240$ the speedup of SEEK and SEIK is not ideal. It is, however, much better than for the mode-decomposed filters. The speedup for SEEK and SEIK reaches respectively 7.6 and 10.6 with 12 processes but is not yet stagnating. The speedup corresponds respectively to a parallel efficiency of 63% and 88%. The speedup of the two filters is smaller for the larger ensemble size since the filter algorithms have been parallelized such that several operations acting on matrices of size $(N-1) \times (N-1)$ remained serial. For the smaller ensemble size, the computation time of these operations was negligible. But, with increasing ensemble size, the execution time of these operations increases strongly, since the complexity of the matrix-matrix operations is proportional to $(N-1)^3$ or $(N-1)^2$. Hence, the execution time for the serial operations can become relevant for larger ensembles. Then, the speedup will be limited by the serial parts according to Amdahl's law.

To exemplify the influence of the serial parts, the resampling phase of SEEK is considered. The execution time and the speedup for the resampling phase of SEEK with $N = 240$ are shown in figure 9.13. The computation of the matrix $\mathbf{T1_p}$ in

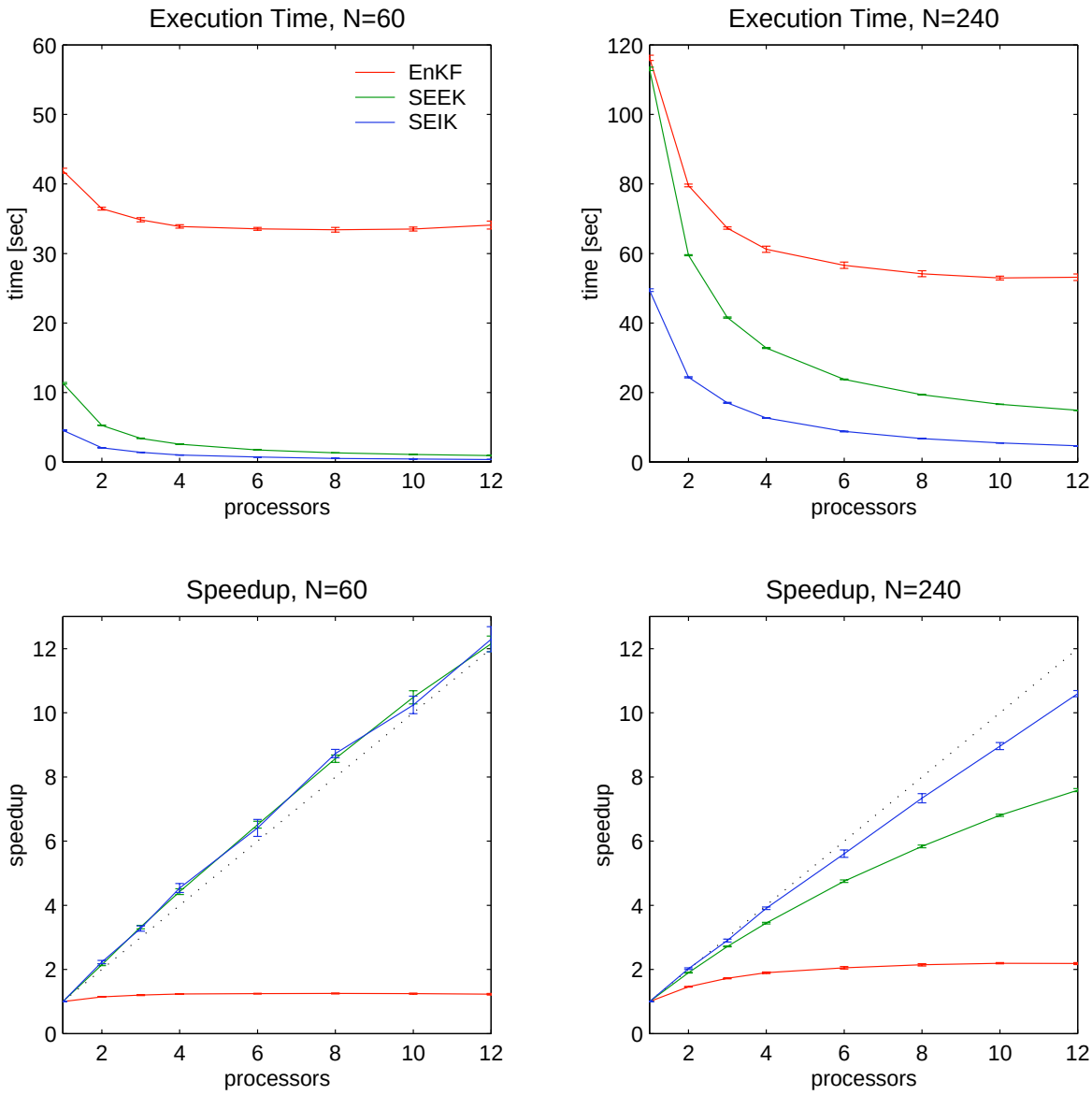


Figure 9.12: Execution time and speedup for the filter update phases for domain-decomposed filter algorithms. Displayed are means and standard deviations analogous to figure 9.10. The left hand side shows results for $N = 60$, the right hand side for $N = 240$. The dotted line shows the ideal speedup.

line 5 of algorithm 7.7 together with the allreduce summation to initialize the global matrix **T1** (line6) shows a slightly super-linear speedup. In addition, an almost ideal speedup is visible for the operations in lines 10 to 14. When the filter is executed by a single process, the operations in lines 5 and 6 together with the operations in lines 10 to 14 take about 95% of the total execution time of the resampling algorithm. Thus, the time for the serial parts of the algorithm is about 5% of the total time. Most of this remaining time is spend in the computation of the singular value decomposition

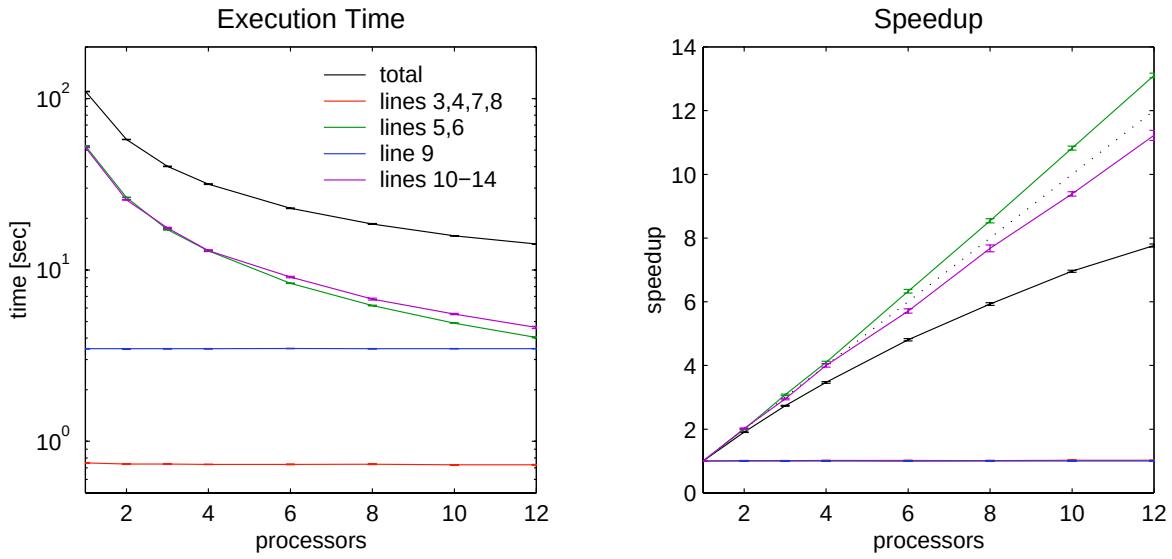


Figure 9.13: Execution time and speedup for the groups of operations in the SEEK resampling algorithm for $N = 240$ for domain-decomposition. Analogous to figure 9.10 means and standard deviations are shown. The line numbers given in the legend of the diagrams refer to those in algorithm 7.7. The dotted line shows the ideal speedup.

of $\mathbf{T1} \in \mathbb{R}^{(N-1)\times(N-1)}$ in line 9. Since this operation is not parallelized, its influence on the total execution time grows with the number of processes. Using 12 processes, the singular value decomposition takes about 25% of the computation time. Thus, the serial parts of the algorithm reduce the parallel efficiency of the resampling algorithm. It reaches only 65% with 12 processes which is consistent with Amdahl's law. The resampling phase dominates the execution time for the full update phase of SEEK. The analysis phase requires only about 6% of the total execution time for the update. Since the efficiency of the analysis algorithm is even minor than that of the resampling algorithm, an efficiency of 63% is obtained for the update phase of SEEK as was mentioned above.

The SEIK algorithm exhibits for $N = 240$ a parallel efficiency superior to the SEEK algorithm. The resampling algorithm of SEIK shows an almost ideal speedup. Its parallel efficiency reaches 95% with 12 processes. The efficiency is influenced by the serial operations in lines 2 to 5 of algorithm 7.10. The efficiency of the full update phase is reduced to 88% by the smaller efficiency of the analysis phase. With a single process, the analysis takes about 15.5% of the total time for the update phase. The efficiency of the analysis phase is limited by serial operations and some communication operations. The most costly serial operation of the analysis phase is the solver step in line 19 of algorithm 7.9. It requires about 6.5% of the execution time for the analysis. There are some other serial and also communication operations like the operation of the matrix $\mathbf{T}$ on some vector (line 20) or the allreduce summation of the matrix **Uinv** in line 11. Together, the serial and communication operations reduce the efficiency of the analysis phase to about 50% with 12 processes.

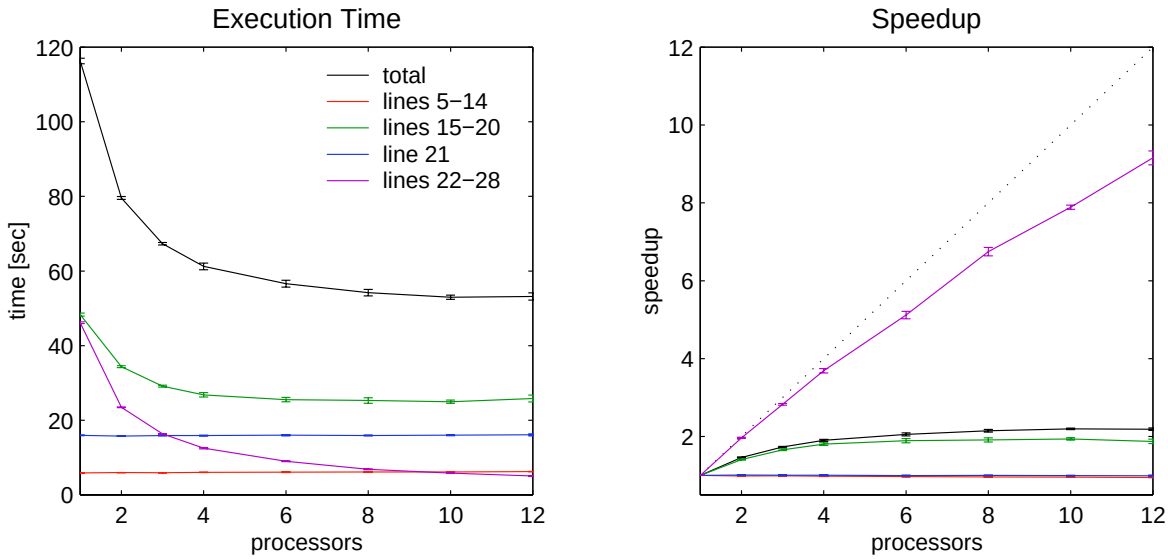


Figure 9.14: Execution time and speedup for the groups of operations in the EnKF analysis algorithm for $N = 240$ for domain-decomposition. Displayed are means and standard deviations as in figure 9.10.The line numbers given in the legend of the diagrams refer to those in algorithm 7.8. The dotted line shows the ideal speedup.

For models of larger dimension n, the influence of the serial operations in the SEIK and SEEK algorithms will be minor. In addition, the amount of communication is independent of the dimension n. Hence, the speedup of the update phases of SEEK an SEIK can be expected to be nearly ideal for larger state dimensions.

The speedup of the domain-decomposed EnKF filter algorithm is very similar to that of the mode-decomposed algorithm. It stagnates at a value of 1.2 for $N = 60$ and 2.2 for $N = 240$.

The reasons for the small speedup are similar to those for the mode-decomposed EnKF. The problem is again exemplified for an ensemble size of $N = 240$. Figure 9.14 shows the execution time and the speedup for operation groups of the domain-decomposed EnKF analogous to figure 9.11. In the domain-decomposed EnKF, the ensemble update with its preparations (lines 22 to 27 in algorithm 7.8) shows a adequate speedup without stagnation. With 12 processes a speedup of 9.1, corresponding to an efficiency of 76%, is reached. The other parts of the algorithm exhibit, however, a much worse speedup. The generation of the observation ensemble together with the initialization of the residual matrix (lines 15-20) requires about 42% of the total execution time if one process is used. For these operations, the speedup stagnates at a value of approximately 2. The operations in lines 5 to 14 are dominated by the computation of **T3** in line 13. This operation is executed serially and requires about 5% of the execution time in the serial case. The solver step for the representer amplitudes **B** in line 21 is not parallelized either. With a single process, it requires approximately 14%

of the total execution time for the EnKF analysis. Overall, a maximal speedup of about 2.2 is obtained for the EnKF analysis algorithm due to the combination of the high amount of serial operations and the small speedup displayed by the generation of the observation.

The speedup achieved for the domain-decomposed EnKF algorithm is even slightly below that for the mode-decomposed algorithm. This is due to the fact that the generation of the observation ensemble exhibits a smaller speedup in the case of domain-decomposition. Additionally, the solver step for the representer amplitudes is serial for domain-decomposition while it is parallelized for mode-decomposition. The routine *Enkf_Obs_Ensemble* is supplied by the user. Case dependent, it might be possible to implement this routine more efficiently. However, even if the time for generating the observation ensemble could be neglected, the total speedup of the EnKF algorithm is limited by the serial operations involving the matrix **T3**. As for the mode-decomposed EnKF algorithm, the speedup will be major if the dimension m of the observation vector relative to the state dimension n is smaller, since the relevance of the serial operations with diminish. This will be, e.g. fulfilled for models of larger state dimension if the amount of observational data remains constant.

9.6 Summary

In this chapter, the parallel filtering framework developed in chapter 8 was implemented and tested with an idealized configuration of the finite element ocean model FEOM. The filtering framework includes the parallel filter algorithms developed in chapter 7.

Data assimilation experiments using synthetic observations of the sea surface height showed a good ability of the filter algorithms to estimate the velocity field. The information provided by surface observations is successfully transported to the lower levels of the model by the estimated covariances between the sea surface height and the velocity field. In contrast to the velocity field, the temperature field is not well estimated. While in the uppermost levels of the model the shape of the true temperature field was accurately estimated, this was not the case for the lower levels. In addition, the temperature was over-estimated in the model levels below a depth of -500 meters.

Experiments assessing the parallel efficiency of the filter framework have been performed with all three ESKF algorithms. The two different process configurations of the framework have been tested. For this, the filter algorithms are either executed by processes which evaluate also the model forecasts or the filter and model parts of the parallel program are executed on disjoint process sets. Both configurations exhibited statistically equal speedups. In addition, the speedup for all three ESKF algorithm is identical within the accuracy of the measurements. The speedup reached a value of about 15 with 18 processes. This corresponds to a parallel efficiency of approximately 83%. The deviation from an optimal parallel efficiency resulted from the fact that different model tasks required slightly different execution times to evaluate the forecasts. This desynchronization yields an overhead in the total execution time which reduces the parallel efficiency.

To assess the speedup of the parallelized filter algorithms, experiments have been performed without time stepping. The experiments included the mode-decomposed and the domain-decomposed filter algorithms. The experiments showed that the model-decomposed SEEK and SEIK filters exhibit a much smaller parallel efficiency than their domain-decomposed counterparts. This is due to a high amount of communication which limits the speedup of the mode-decomposed algorithms. In the experiments the speedup stagnates for the mode-decomposed filters for rather small numbers of processes. The speedup of the domain-decomposed SEEK and SEIK filters did not stagnate for the tested process numbers. For the smaller ensemble size of $N = 60$, the speedup was even super-linear. For the larger ensemble size of $N = 240$, the efficiency of the SEEK and SEIK filters was limited due to serial operations on matrices involving the dimension $r = N - 1$ of the error subspace. The EnKF algorithm exhibited an almost equal parallel efficiency for both parallelization variants. The speedup stagnated at values which are significantly smaller than the speedup obtained with the SEEK and SEIK filters. The limited speedup of the EnKF algorithm is due to serial operations on matrices involving the dimension of the observation vector.

The results for the parallel efficiency obtained in this chapter are specific for the computer system used for the experiments and for the experimental configurations. However, some general conclusions can be drawn. The stagnation of the speedup in the EnKF algorithm will occur independently from the used computing platform if the observation dimension is sufficiently large compared with the ensemble size. The obtained value of the speedup will vary from computer to computer and will depend on the dimensions involved in the data assimilation problem. Similarly one can expect always a decreasing parallel efficiency for the domain-decomposed SEEK and SEIK filters when the ensemble size increases. This is due to serial operations on matrices involving the dimension of the error subspace. The speedup which can be obtained with the mode-decomposed SEEK and SEIK filters is controlled by the ratio of floating point performance to communication performance depending on the computing platform and the dimension of the data assimilation problem.

If the filter framework is used with models of larger state dimension n, a parallel efficiency of the data assimilation system similar to the current experimental results can be expected. In addition, the speedup of the domain-decomposed SEEK and SEIK filters can be expected to be excellent. The speedup of the mode-decomposed variants of these filters will be limited by the high amount of communication which is performed in the algorithms. The speedup of the EnKF algorithms will be limited for both parallelization variants. However, if the state dimension n increases while the amount of observational data remains constant, the speedup of the EnKF algorithms will increase, too.

Chapter 10

Summary and Conclusion

In the second part of this work the application of Error Subspace Kalman Filters (ESKF) on parallel computers was studied. The implementation of the parallel data assimilation system using the ESKF algorithms was conducted in two steps. First, the parallelization of the analysis and resampling phases was discussed. Subsequently, the parallelization of the forecast phase was considered. The latter was included in the development of a framework for parallel filtering. To assess the parallel efficiency of both the filter framework and the parallel filter algorithms, the framework was used to implement a data assimilation system based on the finite element ocean model FEOM. The obtained data assimilation system was tested in experiments with an idealized configuration of FEOM.

With regard to the analysis and resampling phases, the filter algorithms allow for two different parallelization strategies. On the one hand, the ensemble or mode matrix can be decomposed over the processes such that each process holds several columns, i.e. full ensemble states, of the matrix. This strategy is referred to as mode-decomposition. On the other hand, the model domain can be decomposed into sub-domains. Hence, each process holds only the part of a model state which corresponds to its local sub-domain. Using domain-decomposition, the ensemble or mode matrix is decomposed such that each process holds a full ensemble of local sub-states.

The comparison of communication and memory requirements for both parallelization variants showed that the domain-decomposed filters are preferable. The size of communicated matrices is smaller in the case of domain-decomposition. The difference is most significant for the SEEK and SEIK filters. With mode-decomposition, several matrices involving the state dimension n or the dimension m of the observation vector are communicated. In contrast, only communications of matrices involving the typically much smaller dimension r of the error subspace are necessary when the domain-decomposition is applied. In addition, the memory requirements for the domain-decomposed filters are smaller than for the mode-decomposed algorithms. The domain-decomposed variants allow for a better distribution of the large matrices. The memory overhead due to additional matrices which are introduced for the parallelization is also smaller for the domain-decomposed filters. The benefit of the smaller communication requirements with domain-decomposition was confirmed by numerical

experiments. In these, the speedup of the mode-decomposed SEEK and SEIK filters stagnates already for less than 12 processes. The obtained speedup values are below 5. In contrast, no stagnation of the speedup was observed in the experiments applying the domain-decomposed SEEK and SEIK filters.

The EnKF algorithm is problematic concerning communication and memory requirements. With both parallelization strategies, it requires full allocation of matrices involving the dimension m of the observation vector on each process. For large observational data sets, this memory requirement can become critical. Additionally, the EnKF algorithm involves ensemble matrices on the observation space, namely of dimension mN with N being the ensemble size, in communication operations even for the domain-decomposed variant. While for mode-decomposition, the communication requirements of all three filters are of comparable size, the domain-decomposed EnKF algorithms communicate much more data than the domain-decomposed SEEK and SEIK filters. Besides the issue of communication and memory requirements, some operations on matrices involving the dimension m of the observation space are performed serially in EnKF algorithm. In the numerical experiments, the EnKF algorithm exhibited a comparable speedup for both parallelization variants. The speedup stagnated at very small values between 1.2 and 2.4 which was mainly caused by the serial parts of the algorithm.

To obtain a more efficient EnKF algorithm a localized filter analysis was derived. The localization neglects observations beyond some distance from a model sub-domain motivated by the fact that the sampled long-range covariances are in general very noisy. Since, in addition, the true long-range covariances are typically very small, the information content of the sampled long-range covariance is negligible. The localization is, however, an approximation which can cause the model forecasts to become unstable. The localization reduces the effective observation dimension of the analysis algorithm. Hence, the memory as well as the communication requirements of the analysis algorithms are reduced. Accordingly, the parallel efficiency of the algorithm will increase.

A framework for parallel filtering was developed which includes the parallelization of the forecast phase of the filter algorithms. This framework is designed to permit the combination of an existing model with the parallel filter algorithms requiring only minimal changes in the model source code. The framework includes an application program interface. This interface defines the structure of the subroutine-calls which have to be added to the model source code. In addition, the interface to observation-related routines which are called from the filter routines is defined. The organization of the framework uses a clear separation between model and filter routines. In addition, operations related to observations are distributed into separate routines. With this structure, the core routines of the filter algorithms are completely independent of both the model and the observations. For combining the framework with an existing numerical model, the major work will consist in the implementation of the observation-related routines. In addition, routines have to be implemented which perform the model-dependent transition between the state vector required for the filter part and the state fields used in the model.

The framework permits to execute multiple model tasks concurrently. Each of these tasks can be individually parallelized. The required communication of data between filter and model parts of the data assimilation program is performed by the framework. Two different process configurations are supported by the framework. Either the processes which execute the filter routines are also involved in the computation of the model forecasts (denoted as joint process sets) or the filter part of the program is executed on a set of processes which is disjoint from the processes used to compute the model forecasts.

The theoretical examination of the different process configurations showed that none of them is clearly preferable. The configuration with joint process sets permits, on the one hand, to use all processes of the program to compute the model forecasts. In addition, the amount of communication will be smaller than with disjoint process sets. On the other hand, this configuration requires that a matrix holding a sub-ensemble of model states is allocated on one process of each model task. This can increase the memory requirements considerably.

The configuration with disjoint process sets requires only the allocation of a single model state vector on one process of each model task. Further, the possible configurations of the model tasks are more flexible than those for joint process sets. While for joint process sets the sizes of the sub-ensembles which are evolved by the model tasks are to be determined in advance, this is not required for the case of disjoint process sets. Here, the framework sends an ensemble state vector to each idle model task. This technique can be useful if the model tasks have strongly different performances. The number of ensemble members evolved by each model task is dynamically controlled by its performance. The automatic adaption to different performances of the model tasks will, however, only work if ensemble size and performance differences are sufficiently large.

The numerical experiments with FEOM yielded equal speedup values for both process configurations. The speedup was not ideal due to varying execution times of the model forecast on different model tasks. The time required for the analysis and resampling phases of the filters was negligible in these experiments.

Overall, the configuration of the framework with joint process sets should be preferred if the memory requirement of the sub-ensembles on processes which execute also the model is not problematic with the used computer architecture. If memory limitations are too strong, the configuration of the framework with disjoint process sets should be used. This configuration should also be used if there are significant performance differences of the model tasks or if one considers to execute the data assimilation program such that model forecasts are computed concurrently on multiple computers.

Considering the framework and the parallel filters together, the parallelization strategy for the filter routines is independent from the process configuration of the framework. Thus, the framework supports a parallelization strategy on two levels. First, the numerical model and the analysis and resampling phases of the filters can be parallelized

independently. Second, the framework permits to perform the forecast with multiple model tasks which are executed concurrently. In this case, one parallel filter task is coupled with several model tasks by the framework.

The parallelization strategy using mode-decomposition amounts to a parallelization of the filter which is independent from a possible parallelization of the model. In contrast, the strategy using domain-decomposition is most efficient for models which are themselves domain-decomposed. In this case, the decompositions used for the model and the filter should coincide to obtain optimal performance. Distinct decompositions of the domains for model and filter are supported by the framework. They will, however, result in an overhead due to the required reordering of the state information.

Concluding, the study showed that the EnKF algorithm exhibits several problems. These are due to the communication and memory requirements of the filter. In addition, the parallelized EnKF algorithms contain several serial operations on matrices which involve the dimension of the observation vector. If the a large amount of observational data is assimilated, these operations will strongly limit the parallel efficiency of the algorithms. Thus the parallel efficiency of the EnKF algorithm is limited in addition to the inferior serial numerical efficiency in comparison to the SEEK and SEIK filters which has been discussed in part 1 of this work.

The SEEK and SEIK filters show a very good parallel efficiency for domain-decomposed states if the rank r of the approximated state covariance matrix is significantly smaller than the dimension of the observation vector and the state dimension. In this situation, the SEIK filter is the algorithm with the highest parallel efficiency. Using mode-decomposition, the parallel efficiency of both filter algorithms is limited by a large amount of data which has to be communicated by global MPI operations.

The differences between the parallel efficiencies of the analysis and resampling phase of the three ESKF algorithms are less important if the computation time for the forecast phase dominates the full execution time of the data assimilation application. In this case a very good parallel efficiency of the data assimilation system is obtained since the evolution of different model states can be performed independently. The efficiency can be limited by varying execution times for different model tasks. Furthermore serial parts of the program like the model initialization or the output of fields to disk files can be limiting for efficiency.

The parallel filtering experiments showed that the filter framework introduced in this work including the implemented parallel filter algorithms is well suited for realistic large-scale data assimilation applications.

Appendix A

Parallel Computing

A.1 Introduction

This appendix provides an introduction to parallel computing. Section A.2 summarizes the fundamental concepts of parallel computing. Subsequently, in section A.3, quantities for the performance analysis of parallel programs are introduced. In addition, an introduction to the Message Passing Interface (MPI) [27] is given in section A.4. The descriptions summarized here follow those by Foster [22] and Pacheco [59]. Some expressions have been taken from these books.

A.2 Fundamental Concepts

Parallel computing bases on several fundamental concepts and methods. We summarize here the fundamental terms which are used in the main part of this work.

Process
A process can be, intuitively, considered as an instance of a program that is executing more or less autonomously on a physical processor. It is fundamental building block of a parallel program which comprises multiple processes.

Parallelism
Parallelism is the possibility to distribute instructions of some operation over multiple processes to perform the parts of the operation concurrently by the processes. An example is the addition of two vectors $\mathbf{a}$, $\mathbf{b} \in \mathbb{R}^n$. The additions of the components

$$\{\mathbf{c}_i = \mathbf{a}_i + \mathbf{d}_i,\ i = 1, \ldots, n\} \tag{A.1}$$

are mutually independent. Hence, they can be performed concurrently by different processes.

Communication
Communication is the operation to exchange data between different processors. Communications will result in an overhead since the participating processor will not perform

productive work during the communication operation. Communication can be performed either collective or point-to-point. Collective communication involves a group of processes. It is, e.g., used for global summations or broadcast operations. Point-to-point communication operations exchange data between pairs of processes.

Synchronization
Synchronization of the execution of a parallel program is required if the following operations of the program base on the results of previous operations performed by parallel processes. Synchronization yields an overhead which is either due to the required communication or due to processes which idle until the synchronization is completed.

Overhead
The overhead describes the excess of execution time of a parallel program in comparison to a sequential program. The overhead is due to communication, synchronization, and the start-up time of parallel processes.

Granularity
Granularity is the ratio of the time for productive work to the time spent for communication or the start-up of parallel processes. Coarse granularity is obtained if the distributed work consists of a large amount of instructions but only few communications. In this case, the time during which the processors work independently is much larger than the communication time.

Load balancing
To obtain an optimal parallel efficiency of a parallel program, the operational load has to be distributed equally over all processes, denoted as load balancing. Dependent on the problem, the distribution of the operations can either be statically (for regular problems), or dynamically (for irregular or adaptive problems).

Program paradigms
A parallel program paradigm describes the general way in which a program is parallelized. Of the many existing paradigms we describe those two which are the most widely used:
Shared-memory programming utilizes the possibility to use a global address space for the memory of all processes of a parallel program. This can be either achieved by a direct access to all memory locations by all processes or by a virtual global address space of distributed memory. Shared-memory programs can be implemented using the Open-MP standard [57].
Message Passing is used to implement parallel programs on computer systems with distributed-memory. The processes of the parallel program share data by explicitly sending and receiving messages. These communication operations are explicitly implemented, e.g. by calling routines of the Message passing Interface (MPI) [27]. An introduction to MPI is provided in section A.4.

A.3 Performance of Parallel Algorithms

The performance of parallel algorithms can be expressed by several measures which are summarized here.

Performance
The performance of a program is defined as the number of operations performed per time unit. In numerical applications, the performance is usually expressed by floating point operations (flops) per second.

Execution Time
The time that elapses between the startup of the first processor executing a parallel program and the time when the last processor completes execution defines the execution time T of the parallel program.
The execution time will generally depend on the computer being used. I.e., the hardware (processors, memory, network, etc.) as well as the compiler used to generate the program executable will influence the execution time.

Speedup
The speedup $S(p, n)$ of a program which is executed on p processors with some problem size of n is defined by

$$S(p, n) = \frac{T(1, n)}{T(p, n)} \ . \tag{A.2}$$

The speedup describes the factor by which the execution time of a parallel program is reduced with p processors, relative to the execution with a single processor.

Parallel Efficiency
The parallel efficiency $E(p, n)$ measures the process utilization in a parallel program relative to a serial program. It is defined by

$$E(p, n) = \frac{T(1, n)}{p \cdot T(p, n)} \ . \tag{A.3}$$

A parallel efficiency of 1 (or 100%) shows an ideal parallelization. Since the parallel program will not be free of overhead and will usually contain also serial phases, it is $E(p, n) < 1$.

Amdahl's Law
Typically, not all operations in a program can be parallelized. Thus, there will be some fraction α, $(0 \leq \alpha \leq 1)$ of serial operations. The total execution time of a parallel program is then given by the sum of the execution times T_p for the parallel and T_s for the serial fractions of the program:

$$T(p, n) = T_s(1, n) + T_p(p, n) = \left(\alpha + \frac{1 - \alpha}{p}\right) T(1, n) \tag{A.4}$$

Serial parts of a parallel program will limit the speedup, since, according to equation (A.4),

$$S(p,n) = \frac{T(1,n)}{T(p,n)} = \frac{1}{\alpha + \frac{1-\alpha}{p}} \ . \tag{A.5}$$

Thus, the asymptotic speedup is

$$S(p,n) \to \frac{1}{\alpha} \quad \text{for } p \to \infty \ . \tag{A.6}$$

Scalability
A parallel program is scalable if its execution time is inversely proportional to the number of processors used to execute the program. This behavior is denoted as scalability with fixed problem size. Scalability with scaled problem size describes the property of an algorithm to allow for an increase rate of the problem size which keeps the efficiency constant when increasing the number of processors.

A.4 The Message Passing Interface (MPI)

Using the message-passing library MPI the parallel program is written by augmenting standard Fortran or C/C++ source code with calls to library functions for sending and receiving messages.

The MPI-1 standard [27] comprises 129 functions. We describe here fundamental concepts of MPI. In the course of this, we describe the functions which are used for the parallelization of the filter algorithms and for the implementation of the parallel filter framework.

Message Passing
MPI is based on message passing. That is, communication is performed by the explicit sending and receiving of messages which contain the data to be exchanged.

Message
A message consists of the data to be exchanged and an envelope enclosing the message. The envelope contains the information which is necessary to identify a message and to send it to the right process. The identifying information are the rank of the receiving process, the rank of the sending process, a tag, and a communicator. The tag identifies a message if several messages of the same type are sent by the same process.

Initialization of a MPI Program
Before any other MPI functions can be called, the library must be initialized by calling the function *MPI_Init*. After a program has finished using the MPI library, each process must call *MPI_Finalize*. This function ensures a clean termination of MPI, e.g. by freeing memory allocated by the MPI library.

Communicator
A communicator defines a set of processes which can send messages to each other. All communication operations in MPI are performed within a communicator. Accordingly,

a communicator must be specified in the calling interface of all MPI functions which are related to communication or the communicator itself.

The communicator is useful to define subgroups of processes which participate in collective communication operations. After the initialization of a program which is parallelized using MPI, the communicator *MPI_COMM_WORLD* exists which contains all processes of the program. Other communicators can be defined, e.g., by splitting the set of processes in an existing communicator with the function *MPI_Comm_split*.

Rank of a Process

The rank of a process in a communicator is provided by the function *MPI_Comm_rank*. The total size of a communicator in terms of processes is provided by the function *MPI_Comm_size*.

Point-to-Point Communication

The basic point-to-point communication operations of MPI are the functions *MPI_Send* and *MPI_Recv*. These operations are blocking, i.e., a process which calls e.g. *MPI_Recv* remains idle until the message it has to receive is available.

The MPI library provides also non-blocking operations. These are, e.g., the functions *MPI_ISend* and *MPI_IRecv*, which are the non-blocking counterparts of the basic send and receive operations. When a non-blocking function is called, the process posts the communication operation and returns immediately from the function without waiting for the completion of the communication operation. To query the completion of a non-blocking operation, the function *MPI_Test* is called.

Broadcast

A broadcast is a collective operation in which a single process sends the same data to every process of a communicator. The broadcast is conducted by calling the function *MPI_Bcast*.

Reduction

A reduction operation is a collective communication operation in which all processes of a communicator contribute data that is combined using a binary operation. Typical operations are addition or the determination of the maximum value of a variable. The combined result is provided to a single process if the function *MPI_Reduce* is called. If the result of the reduction operation is required by all processes of a communicator, the function *MPI_Allreduce* is called.

Gather

To gather an array which is distributed over the processes of a communicator on a single processor, the function *MPI_Gather* is called. The function *MPI_Allgather* provides the gathered array to all processes.

Barrier

To synchronize the processes, the function *MPI_Barrier* can be called. This function causes each process to block until every process of the communicator has called it.

Appendix B

Documentation of Framework Routines

In this appendix, those routines of the filter framework are documented which have not been shown in the main part of this work. The interfaces of these routines are identical for mode and domain-decomposition The description refers to the variant using mode-decomposition.

Subroutine Next_Observation(*step*,*nsteps*,*time*)
int *step* {Current time step, input}
int *nsteps* {Number of time steps to be computed, output}
real *time* {Current model time, output}

... Initialize *nsteps* and *time* ...

Algorithm B.1: Initialize the number of time steps for the next forecast phase and the current model time. Called from the *Get_State* for joint process sets or the filter main routine for disjoint process sets.

Subroutine Distribute_State(n,$\mathbf{x}$)
int n {State dimension, input}
int $\mathbf{x}(n)$ {State vector to be distributed, input}

... Initialize and distribute model fields ...

Algorithm B.2: Initialize the model fields for a model task from a state vector. Called by *Get_State*.

Subroutine Collect_State(n,$\mathbf{x}$)
int n {State dimension, input}
int $\mathbf{x}(n)$ {State vector to be initialized, output}

... Initialize state vector from model fields ...

Algorithm B.3: Initialize the state vector from the model fields of a model task after a state has been forecasted. Called by *Put_State*.

Subroutine Get_Dim_Obs($step, m$)
int $step$ {current time step, input}
int m {dimension of observation vector, output}

... Initialize m ...

Algorithm B.4: Provide dimension of the observation vector. Called from the filter analysis routines.

Subroutine Measurement($step, m, \mathbf{y}$)
int $step$ {current time step, input}
int m {dimension of observation vector, input}
real $\mathbf{y}(m)$ {observation vector, output}

... Initialize $\mathbf{y}$ ...

Algorithm B.5: Provide the observation vector. Called from the filter analysis routines.

Subroutine Measurement_Ensemble($step, m, N_p, \mathbf{Y_p}, \mathbf{y}$)
int $step$ {current time step, input}
int m {dimension of observation vector, input}
int N_p {local ensemble size, input}
real $\mathbf{Y_p}(m, N_p)$ {matrix holding local observation ensemble, output}
real $\mathbf{y}(m)$ {observation vector, output}

... Initialize $\mathbf{y}$ and $\mathbf{Y_p}$ ...

Algorithm B.6: Provide an ensemble of observations. Called from the EnKF analysis routine.

Subroutine Measurement_Operator($step, n, m, \mathbf{x}, \mathbf{y}$)
int *step* {current time step, input}
int n {state dimension, input}
int m {dimension of observation vector, input}
real $\mathbf{x}(n)$ {state vector, input}
real $\mathbf{y}(m)$ {state vector projected on observation space, output}

... operate with H on $\mathbf{x}$ to obtain $\mathbf{y}$ ...

Algorithm B.7: Implementation of the measurement operator. Called from the filter analysis routines.

Subroutine RinvA(*step*,m,r,$\mathbf{A}$,$\mathbf{B}$)
int *step* {Current time step, input}
int m {Dimension of observation vector, input}
int r {Rank of approx. covariance matrix, input}
real $\mathbf{A}(m, r)$ {Matrix to be multiplied by $\mathbf{R}$, input}
real $\mathbf{B}(m, r)$ {Computed product matrix, output}

... $\mathbf{B} \leftarrow \mathbf{R}^{-1}\mathbf{A}$ ...

Algorithm B.8: Multiply the inverse of the observation error covariance matrix $\mathbf{R}$ with some matrix. Called form the analysis routines of SEEK and SEIK. Since the matrix $\mathbf{A}$ is still required in the algorithms, it must not be modified in the routine.

Subroutine RplusA(*step*,m,$\mathbf{A}$)
int *step* {Current time step, input}
int m {Dimension of observation vector, input}
real $\mathbf{A}(m, m)$ {Input matrix and result of addition, input/output}

... $\mathbf{A} \leftarrow \mathbf{R} + \mathbf{A}$

Algorithm B.9: Add the observation error covariance matrix $\mathbf{R}$ to some matrix. Called by the analysis routine of the EnKF. Since the input matrix $\mathbf{A}$ is not further used in the algorithm, it is overwritten by the sum.

Subroutine Init_Ensemble_SEEK($n, r, \mathbf{x}, \mathbf{Uinv}, \mathbf{V}, status$)
int n {state dimension, input}
int r {rank of approximated covariance matrix, input}
real $\mathbf{x}(n)$ {state estimate, output}
real $\mathbf{Uinv}(r, r)$ {inverse eigenvalue matrix, output}
real $\mathbf{V}(n, r)$ {mode matrix, output}
int *status* {status flag, input/output}

... Initialize $\mathbf{x}$, $\mathbf{Uinv}$, and $\mathbf{V}$ ...

Algorithm B.10: Initialize filter fields for SEEK. Called from filter initialization routines.

Subroutine Init_Ensemble_SEIK($n, N, \mathbf{x}, \mathbf{X}, status$)
int n {state dimension, input}
int N {ensemble size, input}
real $\mathbf{x}(n)$ {state estimate, output}
real $\mathbf{X}(n, N)$ {ensemble matrix, output}
int *status* {status flag, input/output}

... Initialize $\mathbf{x}$ and $\mathbf{X}$ ...

Algorithm B.11: Initialize filter fields for SEIK. Called from filter initialization routines.

Subroutine Init_Ensemble_EnKF($n, N, \mathbf{x}, \mathbf{X}, status$)
int n {state dimension, input}
int N {ensemble size, input}
real $\mathbf{x}(n)$ {state estimate, output}
real $\mathbf{X}(n, N)$ {ensemble matrix, output}
int *status* {status flag, input/output}

... Initialize $\mathbf{x}$ and $\mathbf{X}$ ...

Algorithm B.12: Initialize filter fields for EnKF. Called from filter initialization routines.

Subroutine User_Analysis_SEEK($step, n, r, r_p, m, \mathbf{x}, \mathbf{Uinv}, \mathbf{V_p}$)
int *step* {current time step, input}
int n {state dimension, input}
int r {rank of approximated covariance matrix, input}
int r_p {local rank of approx. covariance matrix, input}
int m {dimension of observation vector, input}
real $\mathbf{x}(n)$ {state estimate, input}
real $\mathbf{Uinv}(r, r)$ {inverse eigenvalue matrix, input}
real $\mathbf{V_p}(n, r_p)$ {mode matrix, input}

... User treatment of filter fields ...

Algorithm B.13: User analysis routine for SEEK. Called from filter main routines. The provided input fields should not be changed.

Subroutine User_Analysis_SEIK($step, n, N, N_p, m, \mathbf{X_p}, \mathbf{x}$)
int $step$ {current time step, input}
int n {state dimension, input}
int N {ensemble size, input}
int N_p {local ensemble size, input}
int m {dimension of observation vector, input}
real $\mathbf{x}(n)$ {state estimate, input}
real $\mathbf{X_p}(n, N_p)$ {ensemble matrix, input}

... User treatment of filter fields ...

Algorithm B.14: User analysis routine for SEIK. Called from filter main routines. The provided input fields should not be changed.

Subroutine User_Analysis_EnKF($step, n, N, N_p, m, \mathbf{X_p}, \mathbf{x}$)
int $step$ {current time step, input}
int n {state dimension, input}
int N {ensemble size, input}
int N_p {local ensemble size, input}
int m {dimension of observation vector, input}
real $\mathbf{x}(n)$ {state estimate, input}
real $\mathbf{X_p}(n, N_p)$ {ensemble matrix, input}

... User treatment of filter fields ...

Algorithm B.15: User analysis routine for EnKF. Called from filter main routines. The provided input fields should not be changed.

Bibliography

[1] J. L. Anderson. An Ensemble Adjustment Kalman Filter for data assimilation. *Mon. Wea. Rev.*, 129:2884–2903, 2001.

[2] J. L. Anderson and S. L. Anderson. A Monte Carlo implementation of the nonlinear filtering problem to produce ensemble assimilations and forecasts. *Mon. Wea. Rev.*, 127:2741–2758, 1999.

[3] A. Bennett. *Inverse Methods in Physical Oceanography*. Cambridge University Press, New York, 1992.

[4] L. Bertino, G. Evensen, and H. Wackernagel. Sequential data assimilation techniques in oceanography. *Int. Stat. Rev.*, 71:223–242, 2003.

[5] C. H. Bishop, B. J. Etherton, and S. J. Majumdar. Adaptive sampling with the Ensemble Transform Kalman Filter. Part I: Theoretical aspects. *Mon. Wea. Rev.*, 129:420–436, 2001.

[6] J.-M. Brankart, C.-E. Testut, P. Brasseur, and J. Verron. Implementation of a multivariate data assimilation scheme for isopycnic coordinate ocean models: Application to a 1993-1996 hindcast of the North Atlantic ocean circulation. *J. Geophys. Res.*, 108(C3):3074, 2003. doi:10.1029/2001JC001198.

[7] K. Brusdal, J. M. Brankart, G. Halberstadt, G. Evensen, P. Brasseur, P. J. van Leeuwen, E. Dombrowsky, and J. Verron. A demonstration of ensemble based assimilation methods with a layered OGCM from the perspective of operational ocean forecasting systems. *J. Mar. Syst.*, 40-41:253–289, 2003.

[8] G. Burgers, P. J. van Leeuwen, and G. Evensen. On the analysis scheme in the Ensemble Kalman Filter. *Mon. Wea. Rev.*, 126:1719–1724, 1998.

[9] V. Carmillet, J.-M. Brankart, P. Brasseur, H. Drange, G. Evensen, and J. Verron. A singular evolutive Extended Kalman filter to assimilate ocean color data in a coupled physical-biochemical model of the North Atlantic ocean. *Ocean Modeling*, 3:167–192, 2001.

[10] B. S. Chua and A. F. Bennett. An inverse ocean modeling system. *Ocean Modeling*, 3:137–165, 2001.

[11] S. E. Cohn. An introduction to estimation theory. *J. Met. Soc. Jpn.*, 75(1B):257–288, 1997.

[12] S. Danilov, G. Kivman, and J. Schröter. A finite-element ocean model: Principles and evaluation. *Ocean Modeling*, 6:125–150, 2004.

[13] Project DIADEM. Development of advanced data assimilation systems for operational monitoring and forecasting of the North Atlantic and nordic seas. URL `http://diadem.nersc.no/index.html`.

[14] F.-X. Le Dimet and O. Talagrand. Variational algorithms for analysis and assimilation of meteorological observations: Theoretical aspects. *Tellus*, 38A:97–110, 1986.

[15] G. Evensen. Open boundary conditions for the Extended Kalman filter with a quasi-geostrophic ocean model. *J. Geophys. Res.*, 98(C9):16529–16546, 1993.

[16] G. Evensen. Inverse methods and data assimilation in nonlinear ocean models. *Physica D*, 77:108–129, 1994.

[17] G. Evensen. Sequential data assimilation with a nonlinear quasi-geostrophic model using Monte Carlo methods to forecast error statistics. *J. Geophys. Res.*, 99(C5):10143–10162, 1994.

[18] G. Evensen. The Ensemble Kalman Filter: Theoretical formulation and practical implementation. *Ocean Dynamics*, 53:343–367, 2003.

[19] G. Evensen and P. J. van Leeuwen. Assimilation of geosat altimeter data for the Agulhas current using the Ensemble Kalman Filter with a quasi-geostrophic model. *Mon. Wea. Rev.*, 124:85–96, 1996.

[20] M. Fisher. Development of a simplified Kalman filter. Technical Memorandum 260, European Centre for Medium-Range Weather Forecasts, 1998.

[21] M. Fisher and E. Andersson. Developments in 4D-Var and Kalman filtering. Technical Memorandum 347, European Centre for Medium-Range Weather Forecasts, 2001.

[22] I. T. Foster. *Designing and Building Parallel Programs.* Addison-Wesley, New York, 1995.

[23] S. Frickenhaus, W. Hiller, and M. Best. FoSSI: Family of simplified solver interfaces for parallel sparse solvers in numerical atmosphere and ocean modeling. *Ocean Modelling*, 2003. submitted.

[24] A. Gelb, editor. *Applied Optimal Estimation.* The MIT Press, Cambridge, 1974.

[25] M. Ghil and P. Malanotte-Rizzoli. Data assimilation in meteorology and oceanography. *Adv. Geophy.*, 33:141–266, 1991.

[26] G. H. Golub and C. F. van Loan. *Matrix Computations*. John Hopkins University Press, Baltimore, 1989.

[27] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI - Portable Parallel Programming with the Message-Passing Interface*. The MIT Press, Cambridge, 1994.

[28] T. M. Hamill, C. Snyder, and R. E. Morss. A comparison of probabilistic forecasts from bred, singular-vector, and perturbed observation ensembles. *Mon. Wea. Rev.*, 128:1835–1851, 2000.

[29] T. M. Hamill, C. Snyder, and J. S. Whitaker. Ensemble forecasts and the properties of flow-dependent analysis-error covariance singular vectors. *Mon. Wea. Rev.*, 131:1741–1758, 2003.

[30] T. M. Hamill and J. S. Whitaker. Distance-dependent filtering of background error covariance estimates in an Ensemble Kalman Filter. *Mon. Wea. Rev.*, 129:2776–1790, 2001.

[31] A. W. Heemink, M. Verlaan, and A. J. Segers. Variance reduced ensemble Kalman filtering. *Mon. Wea. Rev.*, (129):1718–1728, 2001.

[32] I. Hoteit. *Filtres de Kalman Réduits et Efficaces pour l'Assimilation de Données en Océanographie*. PhD thesis, l'Université de Joseph Fourier, Grenoble, France, 2001.

[33] I. Hoteit, D.-T. Pham, and J. Blum. A simplified reduced order Kalman filtering and application to altimetric data assimilation in tropical Pacific. *J. Mar. Syst.*, 36:101–127, 2002.

[34] P. L. Houtekamer and H. L. Mitchell. Data assimilation using an Ensemble Kalman Filter technique. *Mon. Wea. Rev.*, 126:796–811, 1998.

[35] P. L. Houtekamer and H. L. Mitchell. Reply. *Mon. Wea. Rev.*, 127:1378–1379, 1999.

[36] P. L. Houtekamer and H. L. Mitchell. A sequential Ensemble Kalman Filter for atmospheric data assimilation. *Mon. Wea. Rev.*, 129:123–137, 2001.

[37] K. Ide, P. Courtier, M. Ghil, and A. C. Lorenc. Unified notation for data assimilation: Operational, sequential and variational. *J. Meteorol. Soc. Jpn.*, 75(1B):181–189, 1997.

[38] A. H. Jazwinski. *Stochastic Processes and Filtering Theory*. Academic Press, New York, 1970.

[39] S. J. Julier and J. K. Uhlmann. A new extension of the Kalman filter to nonlinear systems. In *Proceedings of AeroSense: The 11th International Symposium on Aerospace/Defense Sensing, Simulation and Controls, Orlando, Florida*, 1997.

[40] S. J. Julier, J. K. Uhlmann, and H. F. Durrant-Whyte. A new approach for filtering nonlinear systems. In *Proceedings of the American Control Conference, Seattle, Washington*, pages 1628–1632, 1995.

[41] R. E. Kalman. A new approach to linear filtering and prediction problems. *Trans. ASME, J. Basic Eng.*, 82:35–45, 1960.

[42] R. E. Kalman and R. S. Bucy. New results in linear filtering and prediction theory. *Trans. ASME, J. Basic Eng.*, 83:95–108, 1961.

[43] G. Karypis and V. Kumar. Parallel threshold-based ILU factorization. Technical Report 96-061, Department of Computer Science, University of Minnesota, 1996.

[44] C. L. Keppenne. Data assimilation into a primitive-equation model with a parallel Ensemble Kalman Filter. *Mon. Wea. Rev.*, 128:1971–1981, 2000.

[45] C. L. Keppenne and M. M. Rienecker. Initial testing of a massively parallel Ensemble Kalman Filter with the Poseidon isopycnal ocean circulation model. *Mon. Wea. Rev.*, 130:2951–2965, 2002.

[46] C. L. Keppenne and M. M. Rienecker. Assimilation of temperature into an isopycnal ocean general circulation model using a parallel Ensemble Kalman Filter. *J. Mar. Syst.*, 40-41:363–380, 2003.

[47] G. A. Kivman. Sequential parameter estimation for stochastic systems. *Nonlin. Proc. Geophys.*, 10:253–259, 2003.

[48] Th. Lagarde, A. Piacentini, and O. Thual. A new representation of data assimilation methods: The PALM flow charting approach. *Q. J. R. Meteorol. Soc.*, 127:189–207, 2001.

[49] P. F. J. Lermusiaux and A. R. Robinson. Data assimilation via Error Subspace Statistical Estimation. part 1: Theory and schemes. *Mon. Wea. Rev.*, 127:1385–1407, 1999.

[50] P. F. J. Lermusiaux and A. R. Robinson. Data assimilation via Error Subspace Statistical Estimation. Part 2: Middle Atlantic bight shelfbreak front simulations and ESSE validation. *Mon. Wea. Rev.*, 127:1408–1432, 1999.

[51] A. C. Lorenc. A global three-dimensional multivariate statistical interpolation scheme. *Mon. Wea. Rev.*, 109:701–721, 1981.

[52] P. M. Lyster, S. E. Cohn, R. Ménard, L.-P. Chang, S.-J. Lin, and R. G. Olsen. Parallel implementation of a Kalman filter for constituent data assimilation. *Mon. Wea. Rev.*, 125(7):1674–1686, 1997.

[53] J. Marotzke, R. Giering, K. Q. Zhang, D. Stammer, C. Hill, and T. Lee. Construction of the adjoint MIT ocean general circulation model and application to Atlantic heat transport sensitivity. *J. Geophys. Res.*, 104(C12):29529–29547, 1999.

[54] Project MERCATOR. URL `http://www.mercator-ocean.fr/`.

[55] R. N. Miller, E. F. Carter Jr., and S. T. Blue. Data assimilation into nonlinear stochastic models. *Tellus*, 51A:167–194, 1999.

[56] H. L. Mitchell, P. L. Houtekamer, and G. Pellerin. Ensemble size, balance, and model-error representation in an Ensemble Kalman Filter. *Mon. Wea. Rev.*, 130:2791–2808, 2002.

[57] OpenMP. URL `http://www.openmp.org/`.

[58] E. Ott, B.R. Hunt, I. Szunyogh, M. Corazza, E. Kalnay, D. J. Patil, and J. A. Yorke. Exploiting local low dimensionality of the atmospheric dynamics for efficient ensemble Kalman filtering. *arXiv:physics/0203058,2002*, 2002.

[59] P. S. Pacheco. *Parallel Programming with MPI.* Morgan Kaufmann, Inc., 1997.

[60] Project PALM. Projet d'assimilation par logiciel multi-methodes. URL `http://www.cerfacs.fr/ palm/`.

[61] D. J. Patil, B. R. Hunt, E. Kalnay, J. A. Yorke, and E. Ott. Local low dimensionality of atmospheric dynamics. *Phys. Rev. Lett.*, 86(26):5878–5881, 2001.

[62] J. Pedlosky. *Geophysical Fluid Dynamics.* Springer, 1979.

[63] T. Penduff, P. Brasseur, C.-E. Testut, B. Barnier, and J. Verron. A four-year eddy-permitting assimilation of sea-surface temperature and altimetric data in the South Atlantic ocean. *J. Mar. Res.*, 60:805–833, 2002.

[64] PETSc. Portable, extensible toolkit for scientific computation. URL `http://www-unix.mcs.anl.gov/petsc/petsc-2/`.

[65] D. T. Pham. A singular evolutive interpolated Kalman filter for data assimilation in oceanography. Technical Report 163, Project IDOPT CNRS-INRIA, 1996.

[66] D. T. Pham. Stochastic methods for sequential data assimilation in strongly nonlinear systems. *Mon. Wea. Rev.*, 129:1194–1207, 2001.

[67] D. T. Pham, J. Verron, and L. Gourdeau. Singular evolutive Kalman filters for data assimilation in oceanography. *C. R. Acad. Sci., Ser. II*, 326(4):255–260, 1998.

[68] D. T. Pham, Jacques Verron, and Marie Christine Roubaud. A singular evolutive extended Kalman filter for data assimilation in oceanography. *J. Mar. Syst.*, 16:323–340, 1998.

[69] F. Rabier, H. Järvinen, E. Klinker, J.-F. Mahfouf, and A. Simmons. The ECMWF operational implementation of four-dimensional variational assimilation. 1: Experimental results with simplified physics. *Quart. J. Roy. Meteor. Soc.*, 126:1143–1170, 2000.

[70] M. Roest and E. Vollebregt. Parallel Kalman filtering for a shallow water flow model. In P. Wilders, A. Ecer, J. Periaux, N. Satofuka, and P. Fox, editors, *Parallel Computational Fluid Dynamics: Practice and Theory, Proceedings of the Parallel CFD 2001 Conference, Egmond ann Zee, Netherlands*, 2002.

[71] R. Sadourny. The dynamics of finite-difference models of the shallow-water equations. *J. Atm. Sci.*, 120:680–689, 1975.

[72] R. Salmon. *Geophysical Fluid Dynamics*. Oxford University Press, 1998.

[73] A. Segers. *Data assimilation in atmospheric chemistry models using Kalman filtering*. PhD thesis, Delft University of Technology, 2002.

[74] A. J. Segers and A. W. Heemink. Parallelization of a large scale Kalman filter: comparison between mode and domain decomposition. In P. Wilders, A. Ecer, J. Periaux, N. Satofuka, and P. Fox, editors, *Parallel Computational Fluid Dynamics: Practice and Theory, Proceedings of the Parallel CFD 2001 Conference, Egmond ann Zee, Netherlands*, 2002.

[75] SESAM. An integrated system of sequential assimilation modules. URL `http://meol715.hmg.inpg.fr/Web/Assimilation/SESAM/`.

[76] D. Stammer, C. Wunsch, R. Giering, C. Eckerts, P. Heimbach, J. Marortzke, A. Adcroft, C.N. Hill, and J. Marshall. The global ocean circulation during 1992-1997, estimated from ocean observations and a general circulation model. *J. Geophys. Res.*, 107(C9):3001, 2002. doi:10.1029/2001JC000888.

[77] Robert F. Stengel. *Optimal Control and Estimation*. Wiley, New York, 1986.

[78] O. Talagrand and P. Courtier. Variational assimilation of meteorological observations with the adjoint vorticity equations: Theory. *Q. J. R. Meteorol. Soc.*, 113:1311–1328, 1987.

[79] M. K. Tippett, J. L. Anderson, C. H. Bishop, T. M. Hamill, and J. S. Whitaker. Ensemble square root filters. *Mon. Wea. Rev.*, 131:1485–1490, 2003.

[80] R. Todling and S. E. Cohn. Suboptimal schemes for atmospheric data assimilation based on the Kalman filter. *Mon. Wea. Rev.*, 122:2530–2557, 1994.

[81] Y. Trémolet and F.-X. Le Dimet. Parallel algorithms for variational data assimilation and coupling models. *Par. Comp.*, 22:657–674, 1996.

[82] Y. Trémolet, F.-X. Le Dimet, and D. Trystram. Parallelization of scientific applications: Data assimilation in meteorology. In *High Performance Computing and Networking, Lecture Notes in Computer Science in Meteorology*. Springer, 1994.

[83] G. Triantafyllou, I. Hoteit, and G. Petihakis. A singular evolutive interpolated Kalman filter for efficient data assimilation in a 3-D complex physical-biogeochemical model of the Cretan sea. *J. Mar. Syst.*, 40-41:213–231, 2003.

[84] P. J. van Leeuwen. Comment on "data assimilation using an Ensemble Kalman Filter technique". *Mon. Wea. Rev.*, 127:1374–1377, 1999.

[85] P. J. van Leeuwen. A variance-minimizing filter for large-scale applications. *Mon. Wea. Rev.*, 131:2071–2084, 2003.

[86] P. J. van Leeuwen and G. Evensen. Data assimilation and inverse methods in terms of a probabilistic formulation. *Mon. Wea. Rev.*, 124:2898–2913, 1996.

[87] M. Verlaan. *Efficient Kalman Filtering Algorithms for Hydrodynamic Models*. PhD thesis, Delft University of Technology, 1998.

[88] M. Verlaan and A. W. Heemink. Reduced rank square root filters for large scale data assimilation problems. In *International Symposium on Assimilation in Meteorology and Oceanography*, pages 247–252. WMO, 1995.

[89] M. Verlaan and A. W. Heemink. Nonlinearity in data assimilation applications: A practical method for analysis. *Mon. Wea. Rev.*, 129:1578–1589, 2001.

[90] J. Verron, L. Gourdeau, D. T. Pham, R. Murtugudde, and A. J. Busalacchi. An extended Kalman filter to assimilate satellite altimeter data into a nonlinear numerical model of the tropical Pacific ocean: Method and validation. *J. Geophys. Res.*, 104(C3):5441–5458, 1999.

[91] A. C. Voorrips, A. W. Heemink, and G. J. Komen. Wave data assimilation with the Kalman filter. *J. Mar. Syst.*, 19:267–291, 1999.

[92] X. Wang and C. H. Bishop. A comparison of breeding and Ensemble Transform Kalman Filter ensemble forecast schemes. *J. Atm. Sci.*, 60:1140–1158, 2003.

[93] M. Wenzel, J. Schröter, and D. Olbers. The annual cycle of the global ocean circulation as determined by 4D VAR data assimilation. *Prog. Ocean.*, 48:73–119, 2001.

[94] J. S. Whitaker and T. M. Hamill. Ensemble data assimilation without perturbed observations. *Mon. Wea. Rev.*, 130:1913–1927, 2002.

[95] A. I. Yaremchuk, M. Yaremchuk, J. Schröter, and M. Losch. Local stability and estimation of uncertainty for inverse problem solvers. *Ocean Dynamics*, 52:71–78, 2001.

Acknowledgments

This work has been prepared and written at the Alfred Wegener Institute for Polar and Marine Research (AWI) advised by Prof. Dr. Wolfgang Hiller and Dr. Jens Schröter. I am grateful for having worked at this institute which provides such superior working conditions for PhD students.

I would like to thank Prof. Dr. Wolfgang Hiller for his guidance and support throughout this work. I am also grateful to Dr. Jens Schröter. We had various stimulating discussions which widely influenced this work. He helped me to keep also an eye on the physical aspects of data assimilation.

During the work, I received support by many persons. In particular, I want to thank Stephan Frickenhaus for his help on parallelization and solver issues. Sergey Danilov supported me finding useful initial conditions for the experiments. He also prepared the idealized configuration of FEOM which I were using.

I wish to thank all members of working group "Scientific Computing", Bernadette, Stephan, Natalja, Meike, and Christian for the friendly working atmosphere. Also Manfred, Dima, Sven, Joana, Verena, Markus, Sergey, and Gennardy deserve thanks for the nice atmosphere during group meetings and other occasions with the inverse-modeling group.

A special thank-you is directed to Gennardy Kivman, Stephan Frickenhaus, and Meike Best for proofreading this thesis. In particular Meike went through the whole text and provided me with numerous remarks.

Thanks to Anja for her understanding and encouragement.