VORtech Computing

the Scientific Software Engineers

P.O. Box 260
2600 AG Delft
The Netherlands

tel. +31(0)15-285 0125
fax. +31(0)15-285 0126
vortech@vortech.nl

Data Assimilation in OpenFOAM using OpenDA

Technical Report TR12-03 versie 1.1

Date

January 8, 2013

Author(s)

J.H. Seubers

By Order of

Vortech B.V.; project A06t

Log-sheet

Version	Author	Date	Description	Review
0.1	JHS	24-04-2012	Draft version	
0.2	JHS	29-06-2012	Release candidate	Mark
1.0	JHS	01-07-2012	Release version	Nils
1.1	JHS	08-01-2013	Update for publication	Nils
File location:		/v3/A06t_OpenFOAM/stage_Henk/report		

Contents

Introduction

Theoretical and experimental aerodynamics are often complementary in the pursuit of solving aerodynamic questions. Both lead to results independently, each method with its own limitations. To combine the two approaches therefore opens up possibilities to overcome the limitations of both. Measurements do not have predicting power and are not available everywhere, whereas simulations represent only a part of reality and the involved uncertainties tend to build up over time. These are just a few examples of problems that can be overcome by *data assimilation*: the incorporation of measurement results into simulation estimates.

This report documents the coupling between OpenDA[1] (a framework for data assimilation) and OpenFOAM[2] (a CFD toolbox). This coupling makes it possible to perform various data assimilation procedures on simulation cases created in OpenFOAM. Two demonstration cases are included that prove the system operational and well performing.

The technical part of this document is divided into three chapters. The use cases and the design overview are given in chapter 1. Chapter 2 treats the mathematical background and justification of the developed models and algorithms. The operation of the system is documented in the user guide (chapter 3), where also the two demonstration cases are treated. The results are summarized and recommendations for future development are discussed in chapter 4.

This report is the result of a 3 month internship at VORtech, in line with the MSc programme of Aerodynamics at the TU Delft. The assignment for this internship can be found in appendix A.

[1] OpenDA is supported by the OpenDA association. See http://www.openda.org
[2] OpenFOAM is made available by the OpenFOAM foundation. See http://www.openfoam.com

Chapter 1

Design

In this chapter, the implementation design of the OpenDA-OpenFOAM coupling is explained. Some cases where data assimilation would be of use in aerodynamic applications are given in section 1.1. Then the design methodology is treated in section 1.2, and finally the implementation overview is given in 1.3.

1.1 Use cases

The most important feature of data assimilation, the possibility to combine the strengths of experiment and simulation, makes it useful in a range of research applications. Possible applications for data assimilation in aerodynamics are:

1. Interpolating (extrapolating) measured quantities in time and space.
2. Extracting meaningful data from noisy measurements.
3. Estimating uncertain terms in governing equations.
4. Separating error sources when performing model validation.
5. Assisting in the selection of measurement locations.

Each of these envisioned applications is interesting in itself, and it would be too much to treat them all in this report. The users guide (chapter 3) has some examples of estimation of uncertain terms.

Interpolation of measurements makes it possible to increase the spatial or temporal resolution of measurement results, if the underlying process is known. In a similar way, estimations can be improved for noisy measurements on a well predictable process. The expected value of the uncertain terms in the process can be determined a posteriori (based on the given

measurements)[1]. For a large enough ensemble, when the covariances may estimated accurately enough, it may even be possible to attribute the error to the different sources of uncertainty, which is useful in performing validation studies. And, since the filter operates on a stochastic process, the uncertainty of the estimates (based on both the measurement and simulation) is known. The effectiveness of possible additional measurement locations on reducing the uncertainties can be quantified.

1.2 Design for verification

During the whole development process, verification of the system was a driving requirement (it was also part of the assignment). It seemed natural to work according to the so called *V model* (figure 1.1), which is based on verification. The first thing that was produced therefore was an acceptance test, in fact the first example case from chapter 3. The analytic solution of this case will be used to verify against.

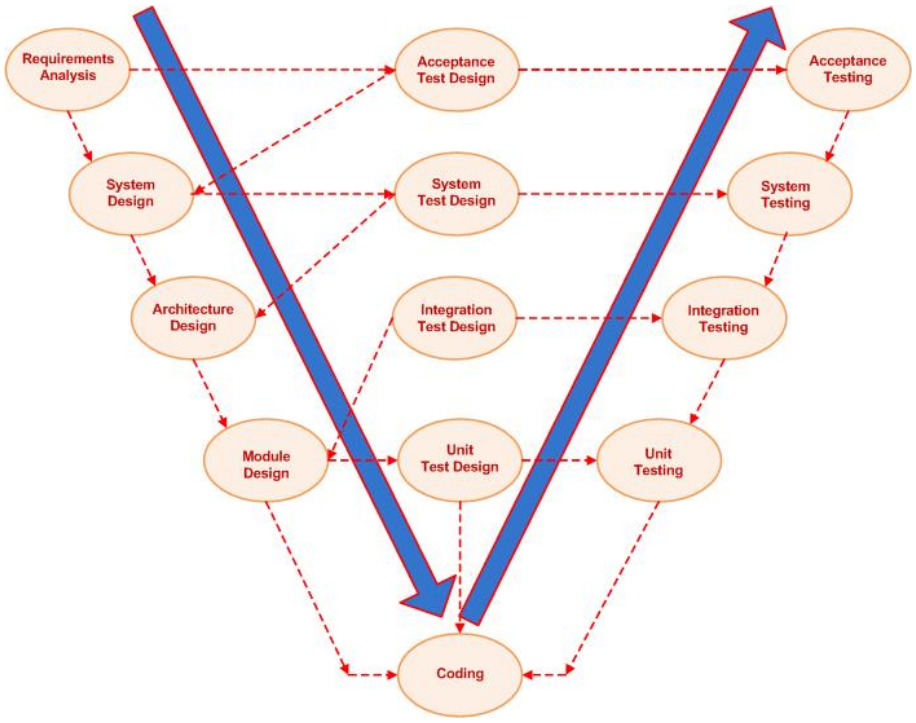

Figure 1.1: V model for software engineering

The overall system design had already been determined by the OpenDA standard. The coupling had to conform to the interfaces defined in this standard[2] and make use of the OpenFOAM system.

[1] This is done by treating the uncertain terms as part of the process state.
[2] Not all of these are very well documented unfortunately

A choice was made to use the black box model from OpenDA, and to implement the coupling using DataObjects to access the OpenFOAM configuration files. However, since the source of OpenFOAM is freely available, it was decided to use the native OpenFOAM classes to read and write these files. However, there was a language barrier to overcome while OpenDA is mostly written in Java, whereas OpenFOAM is written using object oriented C++.

The resulting architecture design was a three layer system. Layer 1 consists of Java classes as part of OpenDA, the only part visible to the user. These classes contain native methods, which form layer 2, or the JNI[3] layer. The methods are implemented as global C++ functions, and perform mostly data translation and memory management. At layer 3, the C++ equivalent of the Java classes are implemented in object oriented style, directly interfacing with the OpenFOAM classes. The class instances in layer 1 and 3 correspond in a 1-to-1 way. This three layer system is illustrated by the process of initializing an OpenFoamDictionary DataObject in figure 1.2.

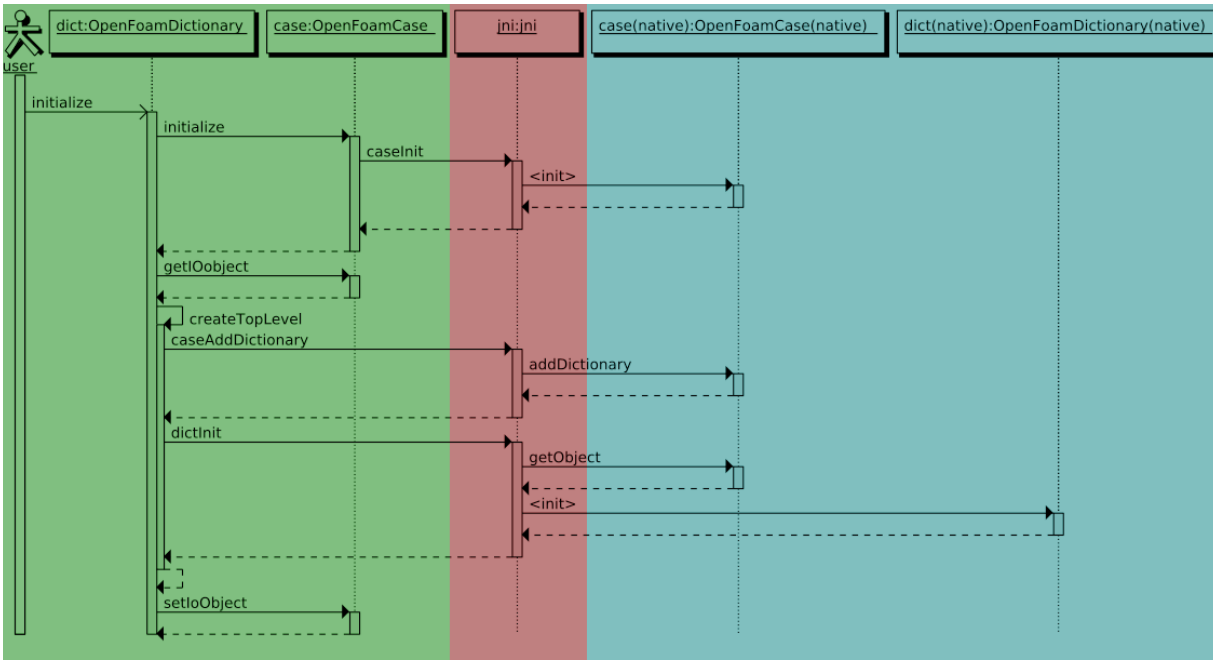

Figure 1.2: Sequence diagram for top-level dictionary initialization

By using this direct native interface, it was possible to gain access to essentially all of the features that are present in OpenFOAM. For example, use has been made of the verbose error reporting of OpenFOAM, and information on the time and grid can be extracted.

For each class that is part of the implementation, unit tests are defined. These test for specific functionality and conformance to the interface. Larger tests are defined that make use of multiple classes, to test their integration. Finally the acceptance test was performed.

[3]JNI: Java Native Interface

1.3 Implementation

The one to one correspondence of the classes is implemented using handles, so that each Java class holds a reference to its native counterpart. An overview of the classes that have been implemented (classes with a $*$ have a native counterpart):

CsvDataObject DataObject to read csv files produced by the OpenFOAM sample utility

OpenFoamCase* Source and registry for DataObjects from a single case directory

OpenFoamDictionary* DataObject to access dictionary files (and sub-dictionaries)

OpenFoamStochDictionary Extension of dictionary that adds configurable random noise

SpatialCorrelationStochVector Noise generator for the different noise models

MeshedGeometryInfo Class that can hold points from any OpenFOAM grid

ObserverUtils Compatibility for the black box model with database-backed observers

The DataObjects are used by the OpenDA black box model to access items (such as fields, parameters, observations boundary conditions) from OpenFOAM. A special object is the OpenFoamCase, which is used to construct all native classes. It lists which DataObjects are available for a specific case directory, and registers those that have been instantiated. For every directory in the file system, there can exist only one OpenFoamCase instance. This is enforced to guarantee uniqueness of all items extracted from a specific OpenFOAM case, which prevents updates from overwriting each other. The same holds for an OpenFoamDictionary class. Only one instance corresponding to a specific dictionary or sub-dictionary can exist per case. This is all invisible to the user, but ensures a proper operation of the system.

The OpenFoamStochDictionary, SpatialCorrelationStochVector and MeshedGeometryInfo operate together to function as a noise model. The OpenFoamStochDictionary reads its configuration and creates a SpatialCorrelationStochVector based upon that. For each ExchangeItem from the OpenFoamDictionary that has an attached MeshedGeometryInfo, the SpatialCorrelationStochVector in turn generates the uncertainties based on the geometry. Two different methods for generating spatial correlated noise have been implemented: a direct method and a spectral method. For more information on the noise models, see section 2.3.

The configuration for each of the classes is documented in appendix C.

Chapter 2

Stochastic modelling

2.1 Terminology

Data assimilation is the *incorporation of measurement results into simulation estimates*. It is a procedure aimed at producing optimal estimates of the *state* of a stochastic process given observations of that process. The stochastic process is realized by the *model* simulation. The observations are measured from a different instance of that process, the *reality*. Note the difference between the two concepts of data assimilation and process calibration.

assimilation Finding the most likely state of a stochastic process

calibration Finding the most likely parametrization of a process

In this report, the *Ensemble Kalman Filter* (EnKF) will be used for data assimilation. Formally, this is an optimal sequential filter for estimating the state of linear processes [1]. It consists of a two stage process, a *forecast* stage and an *analysis* stage. During the forecast, the model is propagated in time from state $\boldsymbol{x}_i^a$ to state $\boldsymbol{x}_{i+1}^f$. An optimal kalman gain $\boldsymbol{K}$ is computed from the uncertainties of the state and the measurements. Then in the analysis stage, observations from the model forecast are generated using the observation operator $\boldsymbol{H}$. These are compared to the the observations from the measurement process $\boldsymbol{y}^o$. The differences between the two, called *innovations*, are then projected back onto the state using the previously computed Kalman gain. It is added to the forecast state $\boldsymbol{x}_{i+1}^f$ to produce the analysis state $\boldsymbol{x}_{i+1}^a$. The next iteration starts from this analysis state. See figure 2.1 for a schematic of the Kalman filter. The equations for the Kalman Filter and the EnKF are given below.

[1] The EnKF is a variant of the Kalman Filter, where an ensemble is used to represent the state covariance. This makes it fit for larger models, and (by approximation) for quasi-linear models as well.

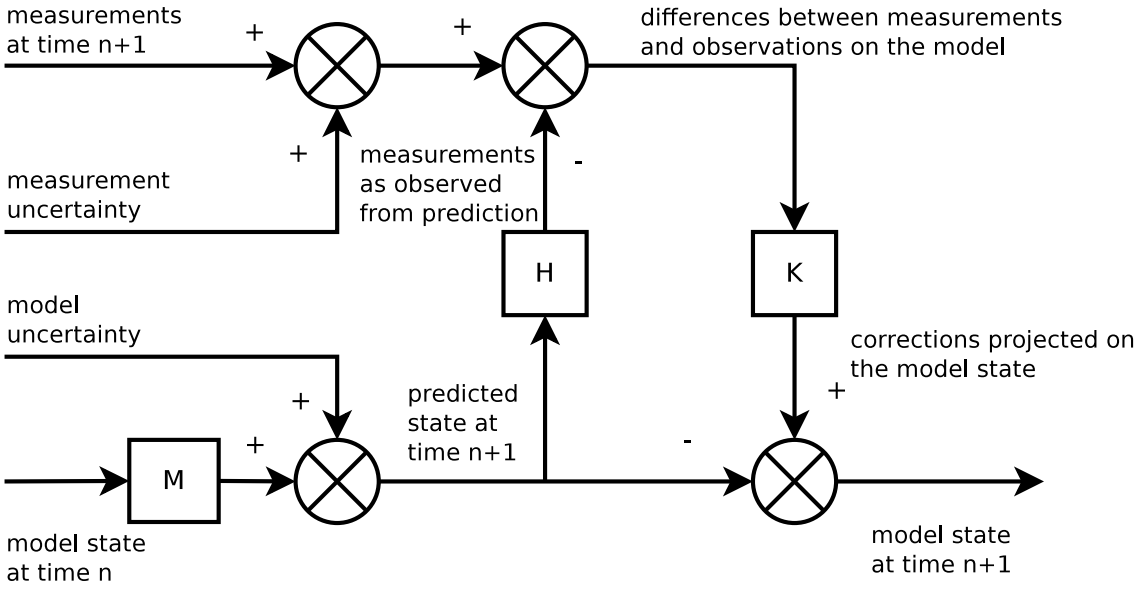

Figure 2.1: Schematic diagram of the Kalman filter

variable	definition
$\boldsymbol{x}$	model state
$\boldsymbol{\eta}$	process noise
$\boldsymbol{\epsilon}$	measurement noise
$\boldsymbol{y}^o$,$\boldsymbol{y}^f$	observations
	(^{o} measured and ^{f} predicted)
N	ensemble size

variable	definition
$\boldsymbol{P}$	covariance of model state
$\boldsymbol{Q}$	covariance of process noise
$\boldsymbol{R}$	covariance of measurement noise
$\boldsymbol{M}$	model time propagation operator
$\boldsymbol{H}$	observation operator
$\boldsymbol{K}$	Kalman gain

Forecast (^{f})

KF	EnKF
$\boldsymbol{x}^f_{i+1} = \boldsymbol{M}_i \boldsymbol{x}^a_i$ $\boldsymbol{P}^f_{i+1} = \boldsymbol{M}_i \boldsymbol{P}^a_i \boldsymbol{M}^T_i + \boldsymbol{Q}_i$	$\boldsymbol{x}^{ef}_{j,i+1} = \boldsymbol{M}_i(\boldsymbol{x}^{ea}_{j,i}) + \boldsymbol{\eta}^e_{j,i}$ $\boldsymbol{x}^f_{i+1} = \frac{1}{N}\sum_{j=1}^{N} \boldsymbol{x}^{ef}_{j,i+1}$ $\boldsymbol{P}^f_{i+1} = \frac{1}{N-1}\sum_{j=1}^{N}(\boldsymbol{x}^{ef}_{i+1} - \boldsymbol{x}^f_{i+1})(\boldsymbol{x}^{ef}_{i+1} - \boldsymbol{x}^f_{i+1})^T$
$\boldsymbol{K}_{i+1} = \boldsymbol{P}^f_{i+1}\boldsymbol{H}^T_{i+1}(\boldsymbol{H}_{i+1}\boldsymbol{P}^f_{i+1}\boldsymbol{H}^{\boldsymbol{T}}{}_{i+1} + \boldsymbol{R}_{i+1})^{-1}$	

Analysis (^{a})

KF	EnKF
$\boldsymbol{x}^a_{i+1} = \boldsymbol{x}^f_{i+1} + \boldsymbol{K}_{i+1}(\boldsymbol{y}^o_{i+1} - \boldsymbol{H}_{i+1}\boldsymbol{x}^f_{i+1})$ $\boldsymbol{P}^a_{i+1} = \boldsymbol{P}^f_i - \boldsymbol{K}_{i+1}\boldsymbol{H}_{i+1}\boldsymbol{P}^f_{i+1}$	$\boldsymbol{x}^{ea}_{i+1} = \boldsymbol{x}^{ef}_{i+1} + \boldsymbol{K}_{i+1}(\boldsymbol{y}^o_{i+1} - \boldsymbol{H}_{i+1}\boldsymbol{x}^{ef}_{i+1} - \boldsymbol{\epsilon}_{j,i})$

2.2 Conservation of flow quantities

In many CFD applications, exact conservation of mass, momentum and energy play a crucial role in the accuracy of simulation results. Finite volume methods, as applied in all OpenFOAM solvers, are able to exactly satisfy the discrete conservation laws, when applied correctly.

It is therefore desirable that the Kalman Filter will preserve this important model property. A conserved quantity q of the model will be called an *invariant*, if its change depends solely upon the forcings. Normalized linear invariants $\boldsymbol{q}$ can be written as an inner product with the model state (see equation 2.1).

$$q_i \equiv \langle \boldsymbol{q}_i, \boldsymbol{x}_i \rangle \tag{2.1}$$

$$\langle \boldsymbol{q}_i, \boldsymbol{q}_i \rangle = c \tag{2.2}$$

Most conserved properties will correspond to a constant $\boldsymbol{q}_i$. For example, if the state $\boldsymbol{x}_i$ consists of average u-velocities in each grid cell, the global u-momentum can be obtained when $\boldsymbol{q}_i$ equals the density (and the inner product is weighted by the grid cell volume in the finite dimensional case. Global mass, global momentum and for instance circulation are all linear quantities that can be represented this way.

Energy quantities can usually be represented as $\boldsymbol{q}_i = G\boldsymbol{x}_i$, but they are not linear in the solution. Therefore, conservation of energy is excluded from the following analysis. A more detailed analysis may reveal if, and under which conditions energy may be preserved. However, since energy is nonlinear, it is present both in the state mean and in the state covariance. Therefore, estimates of energy quantities should deal with the state mean and covariance accordingly.

2.2.1 Conditions on the filter

It can be shown that all linear invariants of the model are also invariants of the model combined with the Kalman filter, provided that they have zero variance. This holds in general for the KF, as well as for the EnKF.

The variance of a linear quantity q in the solution $\boldsymbol{x}$ is

$$\begin{aligned}
\mathrm{Var}(q) &= \mathrm{E}\left[\left(\langle \boldsymbol{q}, \boldsymbol{x} \rangle - \langle \boldsymbol{q}, \mathrm{E}\left(\boldsymbol{x}\right) \rangle\right)^2\right] \\
&= \mathrm{E}\left[\boldsymbol{q}^T(\boldsymbol{x} - \mathrm{E}\left(\boldsymbol{x}\right))(\boldsymbol{x} - \mathrm{E}\left(\boldsymbol{x}\right))^T\boldsymbol{q}\right] \\
&= \boldsymbol{q}^T\boldsymbol{P}\boldsymbol{q}
\end{aligned} \tag{2.3}$$

Given that $\boldsymbol{q}^T\boldsymbol{P}\boldsymbol{q} = 0$ and $\boldsymbol{P}$ is symmetric positive semi-definite, $\boldsymbol{q}$ is a minimizer of the Rayleigh quotient and therefore an eigenvector of $\boldsymbol{P}$ corresponding to the eigenvalue 0. Consequently, the Kalman gain $\boldsymbol{K}$ will be orthogonal to $\boldsymbol{q}$ column-wise (see equation 2.4). There-

fore, the quantity q will not be changed between the forecast and analysis states (2.5).

$$\begin{aligned}\langle \boldsymbol{q}, \boldsymbol{K} \rangle &= \left\langle \boldsymbol{q}, \boldsymbol{P}\boldsymbol{H}^T(\boldsymbol{H}\boldsymbol{P}\boldsymbol{H}^T + \boldsymbol{R})^{-1} \right\rangle \\ &= \left\langle \boldsymbol{H}\boldsymbol{P}\boldsymbol{q}, (\boldsymbol{H}\boldsymbol{P}\boldsymbol{H}^T + \boldsymbol{R})^{-1} \right\rangle \\ &= 0 \end{aligned} \tag{2.4}$$

$$\begin{aligned}\langle \boldsymbol{q}, \boldsymbol{x}^a \rangle &= \langle \boldsymbol{q}, \boldsymbol{x}^f \rangle + \langle \boldsymbol{q}, \boldsymbol{K}(\boldsymbol{y} - \boldsymbol{H}\boldsymbol{x}^f) \rangle \\ &= \langle \boldsymbol{q}, \boldsymbol{x}^f \rangle \end{aligned} \tag{2.5}$$

2.2.2 Conditions on the stochastic model

There are three uncertainties that influence the variance of the conserved quantities.

- The uncertainty of the initial condition $\boldsymbol{\delta x}_0$
- The uncertainty of the model time propagation operator $\boldsymbol{\delta M}$
- The uncertainty of the forcings $\boldsymbol{\delta u}$

In the context of the ensemble EKF, conditions on these uncertainties reduce to conditions on the individual ensemble members. If the model is discretely conservative, an invariant will only change due to the forcings (source terms, boundary conditions etc). Therefore, to render the variance $\mathrm{Var}(q)$ of a conserved quantity zero, it is sufficient to let q be constant over the distribution of the initial condition $\langle \boldsymbol{q}, \boldsymbol{\delta x}_0 \rangle = 0$ and to ensure that the forcing of each ensemble member changes the conserved quantity by the same amount.

2.3 Generating random fields

Producing random numbers can be as simple as rolling a die. Producing random samples from a prescribed distribution can be more involved, but methods exist to generate random scalars from any distribution [2].

Then, generating a random scalar or vector *field* in concept could be as simple as rolling a die at each point. However, producing a random field from a specific distribution [3] or prescribed spatial covariance is more challenging. Here we restrict ourselves to pointwise joint normally distributed fields (which are determined by their spatial covariance). To generate a random field on a given (finite volume) mesh with a prescribed spatial covariance, several methods have been considered.

[2] Some of these methods have been implemented in OpenDA

[3] A probability distribution over fields can be seen as a functional $\mathcal{P} : \{\mathbb{R}^n \to \mathbb{K}^m\} \to \mathbb{R}_+$

- Direct covariance transformation
- Spectral decomposition
- Coarse grid interpolation

The first two methods have been implemented for fields where the covariance only depends on the distance between two points (homogeneous random fields). These methods however are not limited to that case. The third method is more general than the second, but requires a coarse grid. This method could be implemented by using the OpenFOAM multigrid functionality, together with linear interpolation or radial basis functions. This was not finished due to time restrictions.

The direct covariance transformation method requires an explicit covariance matrix between all points of the mesh. Therefore, the distance between each pair of points is calculated. A correlation function is used to compute the correlation coefficients from these distances. A possible correlation function is the Gaussian in equation 2.6. Together with the pointwise standard deviation this defines the covariance matrix.

$$\rho = \exp\left(-\|\boldsymbol{dx}\|^2/\delta^2\right) \tag{2.6}$$

A square root matrix L of the covariance matrix is computed (for instance by Cholesky decomposition). Then each time a realization of the random field is required, a vector of independent standard normally distributed scalars is generated, as many as the number of grid cells. This vector is transformed by the square root matrix into the required field corresponding to the covariance (equation 2.7).

$$\begin{aligned} x &\sim N(0, I) \\ \mathrm{Cov}(Lx, Lx) &= \mathrm{E}\left[(Lx)(Lx)^T\right] \\ &= \mathrm{E}\left[L(xx^T)L^T\right] \\ &= LL^T \end{aligned} \tag{2.7}$$

The spectral decomposition method is based on a decomposition of the field in spectral components. This is implemented on a (n-dimensional) square domain $[0, 1]^n$, where the spectrum is a multidimensional Fourier series (equation 2.10). Here the $\otimes$ symbol signifies a cartesian product in the sequence space, applied once for every dimension > 1.

$$\mathcal{F} : x \mapsto (cos(ikx))_{0 \le i < N} \cup (sin(ikx))_{1 \le i < N} \tag{2.8}$$

$$\boldsymbol{\mathcal{F}} : \boldsymbol{x} \mapsto \prod \mathcal{F}(x_1) \otimes \ldots \otimes \mathcal{F}(x_n) \tag{2.9}$$

$$\eta(\boldsymbol{x}) = \sum_{f_m \in \boldsymbol{\mathcal{F}}} c_m f_m(\boldsymbol{x}) \tag{2.10}$$

This has two advantages: the computational costs can be greatly reduced by reducing the number of variables, and the need to decompose the entire covariance matrix is eliminated.

The fourier coefficients c_m for a limited N can be generated according to any joint normal distribution. The spatial covariance of the field $\eta(\boldsymbol{x})$ will depend on this distribution. Here a simple distribution (2.11) having independent coefficients is chosen.

$$c_m \sim N(0, s_m^2) \qquad (2.11)$$
$$\sigma^2 = 2 \sum_{m=0}^{(2N-1)^n} |s_m|^2$$

The resulting spatial correlation again depends only on distance (equation 2.12). Note this only holds when the c_m are independent and the variances of the sine and cosine parts are equal for the same wave number. Also note that this correlation function is periodic with period $\frac{2\pi}{k}$.

$$\rho = \frac{1}{\sigma^2} \sum_{m=0}^{(2N-1)^n} 2\,|s_m|^2\, cos(k_m x) \qquad (2.12)$$

The disadvantage of the spectral method, is that the covariance functions can only be represented approximately with a limited number of coefficients. Also boundary conditions can only be satisfied if the domain is suitable, and the periodicity must be taken into consideration. A comparison between a correlation function from the direct method, and the corresponding correlation function according to equation 2.12 is given in figure 2.2. A limited number of $N = 6$ modes have been used to capture 95% of the energy in the original correlation function.

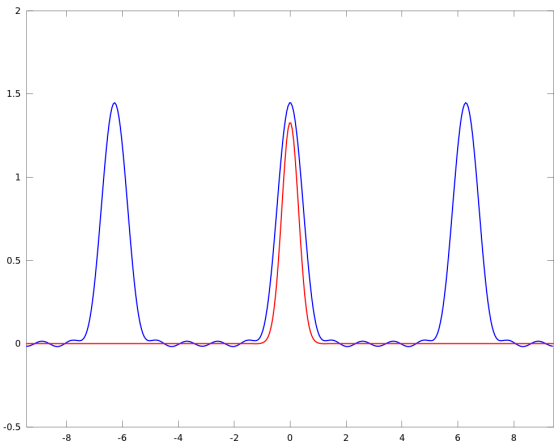

Figure 2.2: Correlation functions for the direct method (red) and the spectral method (blue)

In the current implementation of the spectral method, it is unfortunately not possible yet to configure all coefficients independently. Rather, they are constant up to a limited frequency and scaled to obtain a specified pointwise standard deviation. This can be extended by using a fourier transform of a specified correlation function such as 2.6.

Chapter 3

Users guide

This chapter is intended as a tutorial and reference for anyone intending to perform data-assimilation with OpenFOAM and OpenDA. Its goal is primarily instructional, references to theoretical and technical sections will be given where necessary. First, the central concepts of OpenDA are introduced for OpenFOAM users in section 3.1. These concepts will be applied to two reference problems relevant for aerodynamics, using two tutorials. Section 3.2 presents a 1D advection problem and section 3.3 a 2D incompressible lid driven cavity flow.

3.1 Concepts in OpenDA

Data assimilation is the process of producing estimates of the state of a model based on measurements. To do so any data assimilation system obviously requires a model capable of predicting this state, a set of measurement data, an algorithm to put things together, and a way of presenting its estimates. These are all separate components in OpenDA that interact with each other. The core components that are absolutely necessary for the system to run, are:

- A (stochastic) observer for dealing with the measurements
- A (stochastic) model to provide predictions
- An algorithm to drive the model and the observer and compute estimates

These components can be chosen using configuration files or via the user interface. For example, multiple algorithms exist to choose from, such as an Ensemble Kalman filter or a Particle filter. The algorithm is the most important component, since it tells the other components what to do. In practice, this means that the OpenDA application will run the algorithm, which will run the model in turn. The model in this case will be an OpenFOAM solver, which is driven by OpenDA writing the input and reading the output files. The model

in OpenDA that is responsible for this is called the *black box* model, since it requires no knowledge of the internal workings of the actual model.

Only the input and output formats need to be supported. These are parsed and converted by the black box model, making the contained items availeble to OpenDA. These items are called *exchange items*. In this case, OpenDA actually does know more about the internal workings of OpenFOAM, and this knowledge is in fact used to make geometry data available for the uncertainty models.

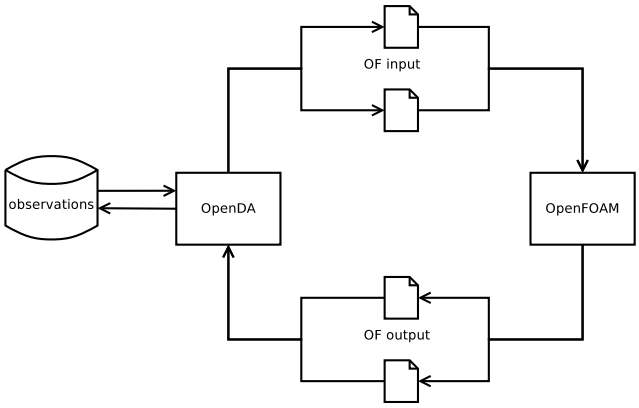

Figure 3.1: Black box model concept

Furthermore, other components can be used and configured in an OpenDA run, including:

- Noise models for creating realizations of uncertainties
- Parallellization models to concurrently evaluate multiple model instances
- Result writers to present the output of the algorithm

Some of these components will be used in the tutorials as well.

3.2 1D periodic advection tutorial

The process of data assimilation will be illustrated by this most simple problem. We know that uniform flow with speed $u_0 = 1$ exists on a periodic domain $x \in [0, 1]$, and that a quantity ϕ is convected with this flow. The domain can be seen as a circular tube, so everything that exits at $x = 1$ enters at $x = 0$. This is our model, as represented by equations 3.2.

$$\phi_t - u_0 \phi_x = 0 \tag{3.1}$$

$$\phi(0, t) = \phi(1, t) \tag{3.2}$$

This system has no unique solution, since the initial condition is unknown. We do however have observations at certaint points x_n^{obs}, let's say a total of N^{obs} evenly spaced points in the domain. We will assimilate these observations to estimate the solution $\phi(x, t)$.

Before we can make an estimate of this process, we must model the uncertainties. Any uncertainty will be approximated by a random variable (scalar or vector), with an assumed distribution (called the prior). We can assume a prior for the initial condition, or a prior for an error source term in the equation, or both. Since the initial condition is unknown, it

would be natural to model it as an uncertainty. However, convergence will then depend on the actual initial condition being within certain bounds, which we can not guarantee. Therefore we will model an error source term in the equation, $\eta(x,t)$. If this term is not too large, the algorithm will converge. The arbitrary initial condition is chosen as zero (since stochastic initialization is not yet supported at this time).

$$\begin{aligned} \phi_t - u_0\phi_x &= \eta(x,t) \\ \phi(x,0) &= 0 \\ \phi(0,t) &= \phi(1,t) \end{aligned} \tag{3.3}$$

3.2.1 Algorithm setup

This is where we select the main components of our data assimilation run, i.e. the algorithm, model and observer. This is done in the main configuration file, located in the tutorial directory `example1d`, named `example1d.oda`. All of the OpenDA configuration files are in XML format. They can be edited using any regular text editor, but use of a validating xml editor (such as XML copy editor or jEdit) is recommended, which is helpful in displaying valid configuration choices [1].

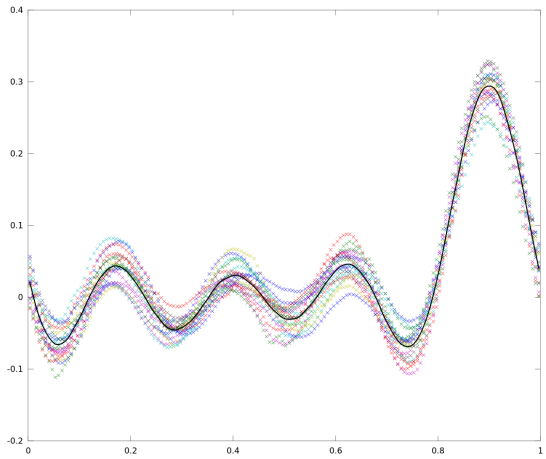

Figure 3.2: Example solution

The content of the main configuration reflects the directory structure of the tutorial, as we will be using a separate directory for each component. This structure, containing the `algorithm`, `model` and `observer` directories, is shown in figure 3.3 (where the top directory represents the OpenDA application component). The configuration for each sub-component is located in its own directory. When there is a dependency, a configuration file referring to another file is indicated by a dashed arrow. To set up this case, we will follow these dashed lines.

The first component that we depend on, is the algorithm. Since this is data assimilation on a small linear problem, the Kalman Filter is the right choice for an algorithm. We will use the Ensemble Kalman filter (EnKF). To select this algorithm, its configuration is given in `example1d.oda` as follows.

```
3    <algorithm className="org.openda.algorithms.kalmanFilter.EnKF">
```

[1] When using a validating XML editor, add the xml schema attributes (xsi:schemaLocation) to the first tag of the file. This will enable validation as well as autocompletion features. Use the schema locations specified in appendix B.

```
    <workingDir>algorithm</workingDir>
    <configFile>enkf.xml</configFile>
  </algorithm>
```

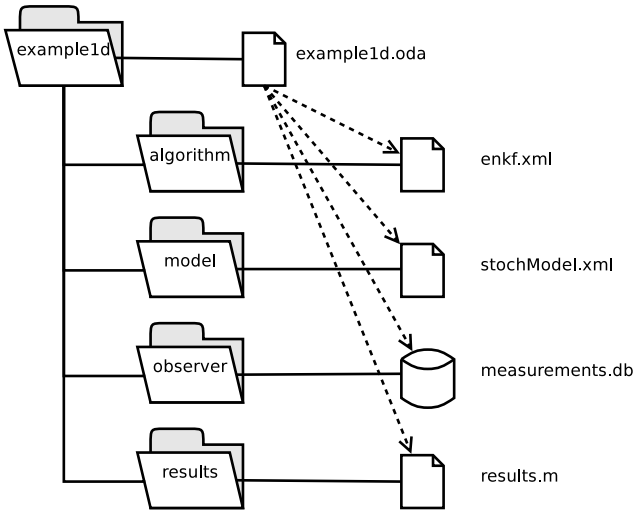

Figure 3.3: Directory structure for the configuration of OpenDA

The configuration file that is referred to here, `algorithm/enkf.xml` then specifies the algorithm options. Each time there are observations available, an analysis step will be done. We will simulate using an ensemble of 20 model instances.

```
<?xml version="1.0" encoding="UTF-8"?>
  <EnkfConfig>
    <analysisTimes type="fromObservationTimes"/>
    <ensembleSize>20</ensembleSize>
    <ensembleModel stochParameter="false" stochForcing="true" stochInit="false"/>
  </EnkfConfig>
```

Now we need an observer to process the measurement data. The (generated) measurement data for this case has been stored in a database. OpenDA includes a database-backed observer component called the Costa observer.

```
    <stochObserver className="org.openda.costa.CtaStochObserver">
      <workingDir>observer</workingDir>
      <configFile>measurements.db</configFile>
    </stochObserver>
```

This database contains exact observations generated from a reference model satisfying the original deterministic equations (3.2). The initial condition for the reference model is $\phi = \text{sinc}(ax - \frac{1}{2})$, with a chosen such to satisfy C^2 continuity at the boundary. [2]

[2] The reference solution can be obtained using the matlab scripts init_condition.m and exact_solution.m in the resources/1d directory.

The configuration also depends on the model that we choose. The `.oda` file that the model component is provided by the *black box* model wrapper. This component is set up using 3 additional config files in the `model` directory, which are explained in section 3.2.2.

```
<stochModelFactory className="org.openda.blackbox.wrapper.BBStochModelFactory">
  <workingDirectory>model</workingDirectory>
  <configFile>stochModel.xml</configFile>
</stochModelFactory>
```

The results of the run will be written in Matlab format in the `results` directory.

3.2.2 Model setup

Now we need an OpenFOAM model for this case. We will use the 'ScalarTransportFoam' solver, which solves the advection-diffusion equation (3.4).

$$\frac{\partial T}{\partial t} + \nabla(\boldsymbol{u} \cdot T) = \nabla(\nu \nabla T) \tag{3.4}$$

The OpenFOAM tutorial case 'pitzDaily' will serve as a template to start our modelling from. A few adaptations have been made to model our case in this solver (see the example files). A 1D mesh has been defined in `blockMeshDict`. Viscosity has been set to $\nu = 0$ in `transportProperties`, and the zero initial conditions and periodic boundary conditions have been specified in the `T` file under the `0` directory. Most importantly, the sample utility of OpenFOAM has been configured to extract measurements from this case, using the dictionary `sampleDict` in the `system` directory. These measurements will be written by the sample utility, and read by OpenDA.

This modified OpenFOAM tutorial is found under the `model` directory, in a directory named `template`. It will be copied once for each ensemble member. The remaining configuration is contained in 3 files in the `model` directory, see figure 3.4. The `stochModel.xml` file specifies which exchange items from the model are part of the state, observations or parameters. In this case, the state is $T(x,t)$ (called `Tfield`), and the observations are T_k^{obs} (called `Tobs`). These exchange items are extracted from the model by so called *DataObject*s or *IoObject*s. For each item, the name of the IoObject to use is specified in the `model.xml` file.

```
<exchangeItems>
  <vector id="startTime" ioObjectId="control"
      elementId="startTime" />
  <vector id="endTime" ioObjectId="control"
      elementId="endTime" />
  <vector id="Tobs" ioObjectId="sample"
      elementId="T" />
  <vector id="Tfield" ioObjectId="Tdict"
      elementId="internalField" />
</exchangeItems>
```

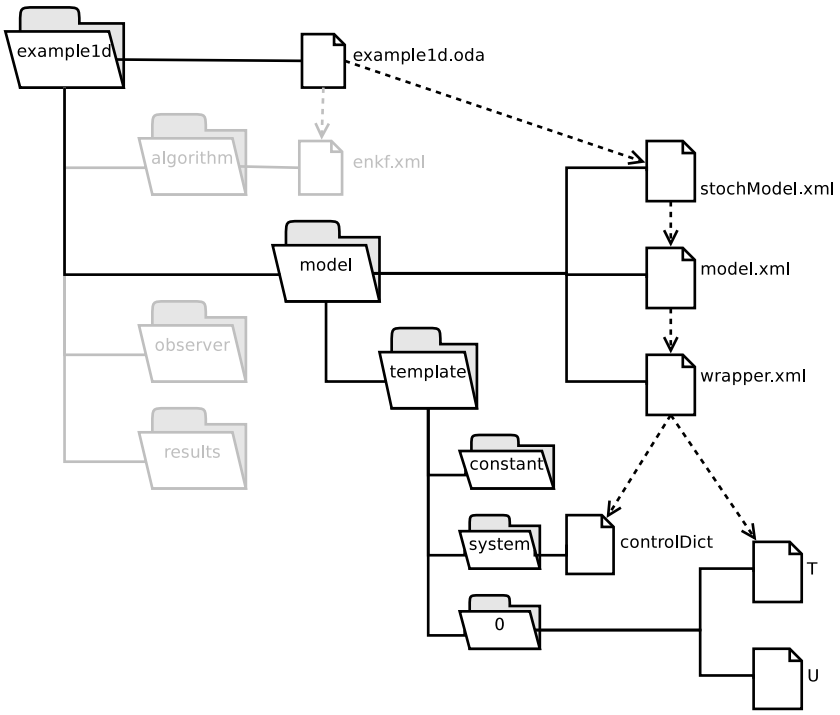

Figure 3.4: Directory structure for the model configuration of the 1D example

There are two special exchange items, `startTime` and `endTime` that allow OpenDA to control the time for OpenFOAM. When OpenDA performs a time step, these items are assigned a value. The time formatting can be controlled by the `<timeFormat>` tag, which must correspond to the OpenFOAM `timeFormat` and `timePrecision` options.[3]

3.2.3 Interaction and I/O

The IoObjects are configured in the `wrapper.xml` file. This case uses 3 IoObjects, called `control`, `sample` and `Tdict`. Each of them is used to access one of the files in the OpenFOAM case. In figure 3.4 this is represented by the dashed arrows pointing from the OpenDA configuration to the OpenFOAM configuration.

The control dictionary requires little setup, only the file name is required. The `OpenFoamDictionary` class reads and writes all files in the OpenFOAM dictionary format[4]

```
<ioObject className="org.openda.blackbox.io.OpenFoamDictionary">
  <file>system/controlDict</file>
  <id>control</id>
</ioObject>
```

[3] See section C.1 for time formatting.
[4] See section C.2 about the OpenFoamDictionary class

The observations generated by the sample utility are in plain csv format, and therefore they are read by the `CsvDataObject` class. This class has more arguments, to specify which columns hold geometry, time and values for the observations.[5]

```
<ioObject className="org.openda.blackbox.io.CsvDataObject">
  <file>sets/%currentTime%/obs_T.csv</file>
  <arg>geometry=x</arg>
  <arg>geometry=y</arg>
  <arg>geometry=z</arg>
  <id>sample</id>
</ioObject>
```

The state can be found in an OpenFOAM dictionary in the current time directory. The current time can be used as an alias in the file name, as in this example. However, since uncertainty is added to the state, this object has more parameters. Grid-dependent noise is generated, using a Fourier expansion. The wave number up to which the expansion will run is given here.[6] Since this noise model can generate noise on a domain of $[0, 1]^d$, and we need a 1-dimensional noise term, we choose $d = 1$ by transforming the 3D grid points to a 1D space using the 1-by-3 `transform` matrix given here.

```
<ioObject className="org.openda.blackbox.io.OpenFoamStochDictionary">
  <file>%currentTime%/T</file>
  <id>Tdict</id>
  <arg>std=0.05</arg>
  <arg>waveNumber=4</arg>
  <arg>transform=[1,0,0]</arg>
</ioObject>
```

Lastly the actions to run the model need to be specified. These are divided into 4 types:

- Initialize (run blockMesh, generate initial conditions)
- Compute (perform a time step using the solver)
- Additional (run the sample utility)
- Finalize

All OpenFOAM executables are run via shell scripts, to set up the OpenFOAM environment settings. Making the OpenFOAM environment global is discouraged, since it interferes with other libraries used by OpenDA. All that is required to run OpenFOAM with OpenDA is making an `~/.OpenFOAM` directory in the user's home, containing a link to the 'etc' directory within the OpenFOAM installation, e.g. `~/.OpenFOAM/2.1.0` links to `/opt/openfoam-2.1.0/etc`. Individual solvers may require setting environment variables within the scripts as well.

[5] See section C.3 about the CsvDataObject class
[6] See section C.4 about the OpenFoamStochDictionary class

3.2.4 Running the algorithm

Now start the run by opening a command prompt in the `example1d` directory and entering '`oda_run.sh example1d.oda`'. This will start the ensemble run and place the output in the `results` directory. To postprocess the results in matlab or octave, use the `load_results.m` script from the `resources/1d` directory. The full ensemble results are in the variable `ensemble`, with dimensions $(N_{\text{cells}}, N_{\text{timesteps}}, N_{\text{ensemble}})$.

- Plot the full ensemble and the ensemble mean at the first few time steps.
- Plot the exact solution alongside (using `exact_solution.m`).
- To what value does the sample standard deviation converge (pointwise average)?
- To what value does the standard deviation centered around the exact solution converge?

The measurement absolute standard error was 0.05 and the standard error of the modelled state uncertainty was also 0.05. The results converge as expected with an uncertainty of $0.05\sqrt{2}$. The measurement uncertainty can be changed by modifying the database `measurements.db` using sqlite. The modelled state uncertainty can be changed in the `wrapper.xml` configuration.

- Perform a run with a different value for the measurement uncertainty or state uncertainty and see if the solution behaves as expected.
- Perform a run with a different ensemble size and note the effects.

Other variables that can be inspected are the observations - variable `obs` $(N_{\text{obs}}, N_{\text{timesteps}})$ - and the Kalman gain for each observation - variable `kg` $(N_{\text{cells}}, N_{\text{timesteps}}, N_{\text{obs}})$.

3.3 2D lid driven cavity tutorial

In this tutorial a more involved example based on the OpenFOAM lid driven cavity tutorial will be under investigation. We have a database with observations of the vertical velocity along the centerline which we will use for data assimilation.

We will solve the problem of estimating the state of the cavity and its boundary condition at the top based on the measured data. The deterministic model for this case will be the steady state incompressible Navier Stokes equations with boundary conditions as given by 3.5. They are solved using the `icoFoam` solver, by time stepping towards the steady state solution.

$$\left.\begin{aligned} \nabla \cdot \boldsymbol{u} &= 0 \\ \nabla p &= \nabla \cdot \left(-\boldsymbol{u} \otimes \boldsymbol{u} + \tfrac{1}{Re}\left(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T\right)\right) \end{aligned}\right\} \quad \text{on } \Omega \tag{3.5}$$

$$\left.\begin{aligned} \boldsymbol{n} \cdot \nabla p &= 0 \\ \boldsymbol{u} &= \boldsymbol{0} \end{aligned}\right\} \quad \text{on } \partial\Omega_{\text{left,bottom,right}} \tag{3.6}$$

The boundary condition at the top is unknown, therefore we will introduce the uncertainty model 3.7. This is an AR(1) process that models the boundary condition as a random departure from the previous estimate, with a certain spatial correlation. As an initial condition the velocity of the moving wall is assumed identically zero, so that the cavity will be at rest. The actual state of the moving wall will follow from assimilation of the measurement data.

$$\left.\begin{array}{ll} \boldsymbol{n}\cdot\nabla p & = 0 \\ \boldsymbol{u}_0 & = \boldsymbol{0} \\ \boldsymbol{t}\cdot\boldsymbol{u}_{j+1} & = \alpha\boldsymbol{t}\cdot\boldsymbol{u}_j + \eta(x) \end{array}\right\} \text{on } \partial\Omega_{\text{top}} \tag{3.7}$$

3.3.1 Algorithm setup

In OpenDA, as with OpenFOAM, it is easiest to start from an existing case. Therefore copy everything from the 1D tutorial to a new directory called `example2d`. Since the same components are used, the `example1d.oda` file remains unchanged (rename it to `example2d.oda`). The algorithm configuration (EnKF) also remains unchanged, however some modifications to the model and observer components will be necessary. We start by configuring the observer for the new data to be assimilated. These are samples from the calculation of a similar case, where the velocity of the top wall was prescribed. The recorded vertical velocities at the centerline of the cavity are stored in a database (located in the `resources/2d` directory).

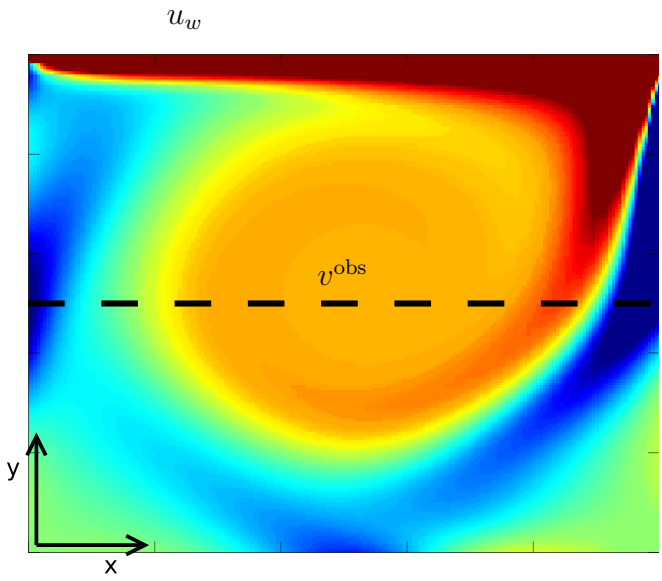

Figure 3.5: Cavity flow at $Re = 1000$

- Copy the database `measurements.db` to the `observer` directory.
- Extract the observation locations from the database using

```
sqlite3 -separator ' ' measurements.db \
          'select "(",xpos,0.05,0.005,")" from stations'
```

3.3.2 Model setup

The OpenFOAM model for this example will be based upon the `cavity` tutorial from OpenFOAM. The whole case directory can be copied to the `model` directory. A few adaptations have to be made:

- Let the kinematic viscosity $\nu = 1 \cdot 10^{-4}$ so that $Re = 1000$.
- Set the time step to $dt = 0.005$ seconds and the time precision to 3 decimals (fixed).
- Rename the `0` directory to `0.000` to make the time names consistent.
- Change the initial velocity of the moving wall to $(0, 0, 0)$.
- Copy the `sampleDict` dictionary from the 1D example (`template` directory)
- Plug in the new observation locations in the `sampleDict`.

Now some configuration changes in the OpenDA black box model are required. We change the solver from `scalarTransportFoam` to `icoFoam` by changing the solver alias in `model.xml`. The template directory should also be changed here from `template` to `cavity`. The state and observation variables have changed as well. The state formally consists of the velocity and pressure fields, and the uncertainty on the boundary conditions (3.8). The observations are once again generated by the sample utility, but now it is the vertical velocity component that we are interested in.

$$\text{state} = \begin{pmatrix} u(x,y) \\ v(x,y) \\ p(x,y) \\ u_w(x) + \eta(x) \end{pmatrix} \tag{3.8}$$

This translates into the exchange items for the state (`internalUfield`, `internalpField` and `boundaryUField`), and for the observations (`Uobs`) respectively. These are configured in the `model.xml` file.

```
<vector id="Uobs" ioObjectId="sample"
   elementId="U_1" />
<vector id="internalUField" ioObjectId="UField"
   elementId="internalField" />
<vector id="boundaryUField" ioObjectId="boundaryUField"
   elementId="value" />
<vector id="internalpField" ioObjectId="pField"
   elementId="internalField" />
```

3.3.3 Interaction and I/O

Finally the necessary IoObjects are configured in the `wrapper.xml` file. A total of 5 IoObjects are required (`control`, `sample`, `UField`, `pField`, `boundaryUfield`). The `sample` object requires only a change of filename, since we are observing velocity U instead of the scalar T (use `obs_U.csv` instead of `obs_T.csv`). The `UField` and `pField` are straightforward instances of the `OpenFoamDictionary` class (use filenames `%currentTime%/U` and `%currentTime%/p`, respectively).

The `boundaryUField` however, needs to access the boundary condition for the velocity field. OpenFoam expects this in a nested sub-dictionary in the velocity dictionary file, as can be seen in the fragment below.

```
boundaryField
{
    movingWall
    {
        type fixedValue;
        value ...
```

A sub-dictionary is treated by OpenDA in the same way as a regular file, only the title should be given as an additional argument to the `OpenFoamDictionary` class, right after the file name. In this case, the required value is two levels deep, so the two added arguments are 'boundaryField', then 'movingWall'. See the extract from `wrapper.xml` below. The uncertainty model parameters are different, because the noise component $\eta(x)$ is now generated using a direct noise model, where the spatial correlation is prescribed. The argument is a characteristic correlation length. The component of the vector field, to which the noise is added, is given by the `components` matrix. When multiple independent noise components are desired, add multiple row vectors with different directions in this matrix, separated by a semicolon (`;`).

```
<ioObject className="org.openda.blackbox.io.OpenFoamStochDictionary">
    <file>%currentTime%/U</file>
    <id>boundaryUField</id>
    <arg>boundaryField</arg>
    <arg>movingWall</arg>
    <arg>std=0.05</arg>
    <arg>length=0.02</arg>
    <arg>components=[1,0,0]</arg>
</ioObject>
```

3.3.4 Running the algorithm

Start the run by executing '`oda_run.sh example2d.oda`' from the `example2d` directory. This could take some time, depending on your processor and disk performance. The size of the results file will be quite large. When ready, start postprocessing with matlab or octave, using the `load_results.m` script and other scripts from the `resources/2d` directory. It is advisable to save the results in binary format (type `save results.dat`) to save space and time for later postprocessing. The full ensemble results are in the variable `ensemble` ($N_{\text{cells}}, N_{\text{timesteps}}, N_{\text{ensemble}}$). The grid for this case is in the variables `X,Y` ($2N, 2N$).

- Extract the fields from the ensemble state at time level i by typing
  `[U,V,W,p,Ub,Vb,Wb] = getstate(ensemble(:,i,:) N);`
- Plot these fields using the `imagesc` command.

- Plot some measure of the total state uncertainty over time.

The results converge, and we can deduce that the most likely flow state in the reference case (from which the measurements originate), was the cavity flow resulting from a left-moving lid.

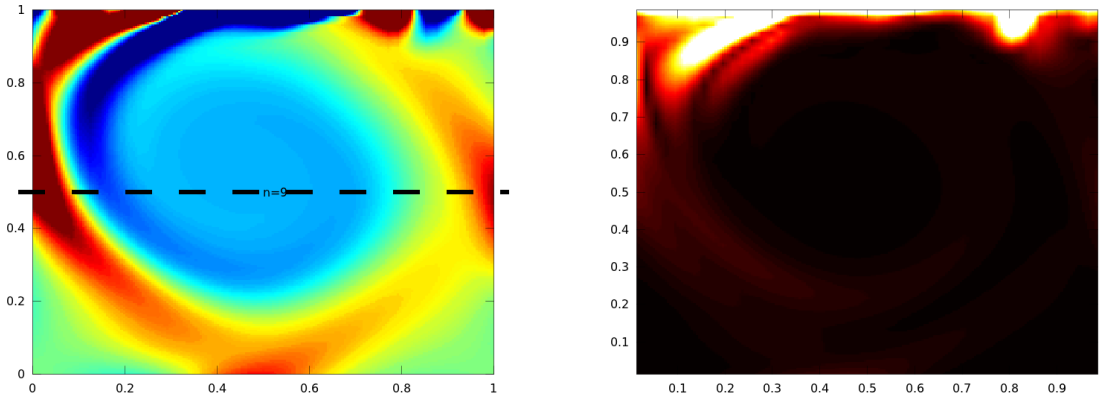

Figure 3.6: Vorticity of the expected velocity, and its pointwise standard deviation

Chapter 4

Conclusions and Recommendations

OpenDA provides data assimilation concepts and procedures to OpenFOAM. These are readily usable with any OpenFOAM solver, applicable to any field variable, parameter or observation. The functionality and performance of the system has been proven using two test cases. A native interface to the OpenFOAM classes was created to access OpenFOAM functionality. Grid geometry information from OpenFOAM is available to OpenDA for use in noise modelling and data interpretation. A direct and a spectral uncertainty model have been implemented for arbitrary unstructured meshes. Small time scales and large state sizes are supported. Detailed error reporting is incorporated.

This functionality could be extended to include

- Better configurability of the spectral noise model (e.g. arbitrary correlation functions).
- Noise models with arbitrary constraints (necessary for mass conserving boundary conditions).
- Support for correlation between different field quantities and correlation in time.
- Support for multigrid facilities from OpenFOAM.
- Model calibration of time-dependent models.

Furthermore, existing functionality can be tested by investigation of more advanced model problems. Questions on the observability of aerodynamic processes can be answered. The effects of ensemble size or different data assimilation algorithms can be examined.

Appendix A

Assignment

Background

Experimental flow data, acquired via particle image velocimetry (PIV), are often incomplete, in the sense that measurements in some locations are unreliable or unavailable. As the circumstances causing the loss of data cannot always be taken away, the missing observations require treatment. To estimate the most likely values for these observations, data assimilation and flow modelling procedures need to be combined.

Assignment

To be able to solve this problem in a generic way, existing solutions are to be put together: OpenDA, a framework for data assimilation, and OpenFOAM, a CFD toolbox. The coupling between these software packages needs to be implemented at the source level. The intended use of this work is data filling in PIV results, by applying ensemble Kalman filtering and Navier-Stokes modelling to regions with missing observations.

To validate the solution, a series of test cases and procedures needs to be defined, implemented, and evaluated. Possible test cases range from a simple 1-d convection-diffusion problem to 2-d Navier-Stokes vortex transport in uniform flow, with appropriate stochastically generated data for the boundary conditions. Using these test-cases, the solution can be verified by comparing the results to a known solution.

Furthermore, for the cases considered, the tractablity of applying various flow models and data assimilation methods can be evaluated, together with the accuracy of the resulting estimates. User documentation should be provided, together with any resultant recommendations.

Project goals

The purpose of this project is to provide a coupling between OpenDA and OpenFOAM, and to demonstrate the results on a test case.

Work breakdown

Short literature study on data assimilation and Kalman filtering	40 hours
Install and configure OpenDA and OpenFOAM	25 hours
Assemble 1-d and 2-d steady/unsteady test cases in OpenFOAM	40 hours
Implement coupling between OpenDA and OpenFOAM	100 hours
Generate statistical test setup from a known solution	20 hours
Verify the solution using the test cases	25 hours
Asses the influence of process parameters on reliability of estimates	24 hours
Provide user documentation for using OpenFOAM with OpenDA	40 hours
Evaluate (if time allows) performance of different combinations of methods	40 hours
Report results and recommendations	60 hours

Total time based on estimate: 414 hours
Total time based on 40hr/wk: 480 hours
Total time based on 18 ECTS: 504 hours

Appendix B

Configuration schemas

example1d.oda, example2d.oda

```
<?xml version="1.0" encoding="UTF-8"?>
<openDaApplication xmlns="http://www.openda.org"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.openda.org
      http://schemas.openda.org/openDaApplication.xsd">
```

enkf.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<EnkfConfig xmlns="http://www.openda.org"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.openda.org
    http://schemas.openda.org/algorithm/enkf.xsd">
```

stochModel.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<blackBoxStochModel xmlns="http://www.openda.org"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.openda.org
    http://schemas.openda.org/blackBoxStochModelConfig.xsd">
```

model.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<blackBoxModelConfig xmlns="http://www.openda.org"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.openda.org
    http://schemas.openda.org/blackBoxModelConfig.xsd">
```

wrapper.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<blackBoxWrapperConfig xmlns="http://www.openda.org"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.openda.org
    http://schemas.openda.org/blackBoxWrapperConfig.xsd">
```

Appendix C

Options reference

This appendix contains an overview of the configuration options that come with the OpenFOAM-OpenDA coupling. On how to use these options, see the tutorial sections 3.2 and 3.3.

C.1 Time formatting options

configuration files: stochModel.xml, system/controlDict

Both OpenFOAM and OpenDA are aware of the model time, i.e. the time that passes in the simulated process. Therefore, they should interpret time values wherever they occur in the same way. Unfortunately the default conventions on time formatting in OpenFOAM and OpenDA are not compatible (OpenFOAM uses a decimal or scientific notation in seconds, OpenDA uses a modified julian date or a date-time string). However, the time format used by the OpenDA black box model (for *exchange items*, alias values etc) can be changed. Use the `<timeFormat>` option in the black box configuration file `stochModel.xml`. The syntax is given in table C.1

Option format	possible value	effect
`<timeFormat>`*string* `</timeFormat>`	`%`x`.`y`f`	Time values are interpreted as seconds in fixed decimal representation, with x significand places before and y after the decimal point.
	`yyyy-MM-dd'T'HH:mm:ss`	The default date-time format. For the specification, see SimpleDateFormat.

Table C.1: Option syntax for time formatting

In OpenFOAM, make sure to select the `timeFormat fixed` option (in the controlDict). Then specify for the `timePrecision` option the same number of significand places *after* the decimal sign as in OpenDA.

C.2 OpenFoamDictionary options

configuration file: wrapper.xml	class: org.openda.blackbox.io.OpenFoamDictionary

To set up an IoObject to access an OpenFOAM dictionary, use the OpenFoamDictionary class. Its options are specified in table C.2 Note: a sub-dictionary is a part of an OpenFOAM dictionary file within curly brackets ({}), containing a group of entries.

Option format	value	effect
`<id>`*string*`</id>`		Specify a name for referring to this IoObject.
`<file>`*path*`</file>`		Specify the relative path with respect to the case directory of the OpenFOAM dictionary file to access.
`<arg>`*entry*`</arg>`	(optional)	Specify the entry name of a sub-dictionary within the dictionary file. This works recursively: specify additional arguments to open nested sub-dictionaries.

Table C.2: Option syntax for OpenFoamDictionary IoObject

C.3 CsvDataObject options

configuration file: wrapper.xml	class: org.openda.blackbox.io.CsvDataObject

The CsvDataObject is used to extract data from Csv files. It converts (groups of) columns to exchange items, which can be used as a (read-only) part of the state or observations. Grouped columns are stored in row-major order. Geometry or time info will be added to the exchange items, if specified. The options to this IoObject are given in table C.3.

Option format	value	effect
`<id>`*string*`</id>`		Specify a name for referring to this IoObject.
`<file>`*path*`</file>`		Specify the relative path with respect to the case directory of the csv file to access.
`<arg>geometry=`*name*`</arg>`	(optional)	Specify the column name of a column containing geometry (coordinate values). Groups geometry columns when used multiple times.
`<arg>time=`*name*`</arg>`	(optional)	Specify the column name of a column containing time values.
`<arg>value=`*name*`</arg>`	(optional)	Specify the column name of a column containing (observed) values. Groups value columns when used multiple times.

Table C.3: Option syntax for CsvDataObject IoObject

C.4 Noise model options

configuration file: wrapper.xml	class: org.openda.blackbox.io.OpenFoamStochDictionary

These are the options to the noise model. They must at this time be given as additional options to the OpenFOAMStochDictionary IoObject, following the OpenFOAMDictionary options as described in C.2. This is because the OpenDA black box model does not support autonomous noise models yet. There are two noise models available this way, a direct model and a spectral model (see section 2.3).

Note that the noise can be generated on a transformed domain using a transformation matrix M (from the model domain to the noise domain). This has two advantages:

- The dimensionality of the noise domain can be reduced.
- Scaling and directionality changes can be made easily.

The first property can lower the computational costs of noise generation drastically. It is achieved by using a n-by-m transformation matrix, where n is the dimension of the noise domain and m is the dimension of the model or physical domain. The second property is especially useful for the spectral model, which generates noise on a fixed domain $[0, 1]^n$.

Option format	value	effect
`<arg>std=`*real*`</arg>`		Specify the pointwise standard deviation for the stochastic component that is added to this field.
`<arg>length=`*real*`</arg>`		Select the direct noise model. Specify the correlation length δ to define the spatial correlation coefficient $\rho = \exp\left(-\|\boldsymbol{dx}\|^2/\delta^2\right)$
`<arg>waveNumber=`*integer*`</arg>`		Select the spectral noise model. Specify the cutoff wave number used in the Fourier expansion of the stochastic component. This is uniform in all directions.
`<arg>transform=`*matrix*`</arg>`	(optional)	Specify the transformation matrix M from the model domain to the noise domain.
`<arg>components=`*matrix*`</arg>`	(optional)	Specify how the generated noise fields should be mapped to vector components. For each row of this matrix, an independent stochastic field is generated and mapped to the vector component corresponding to that row.
`<arg>mesh=`*name*`</arg>`	(optional)	Specify the name of the mesh used for the geometry. This can be the name of a patch for boundary fields.

Table C.4: Option syntax for the noise model

OpenDA applications

OpenDA

Open source software environment for data assimilation

Data assimilation for OpenFOAM®

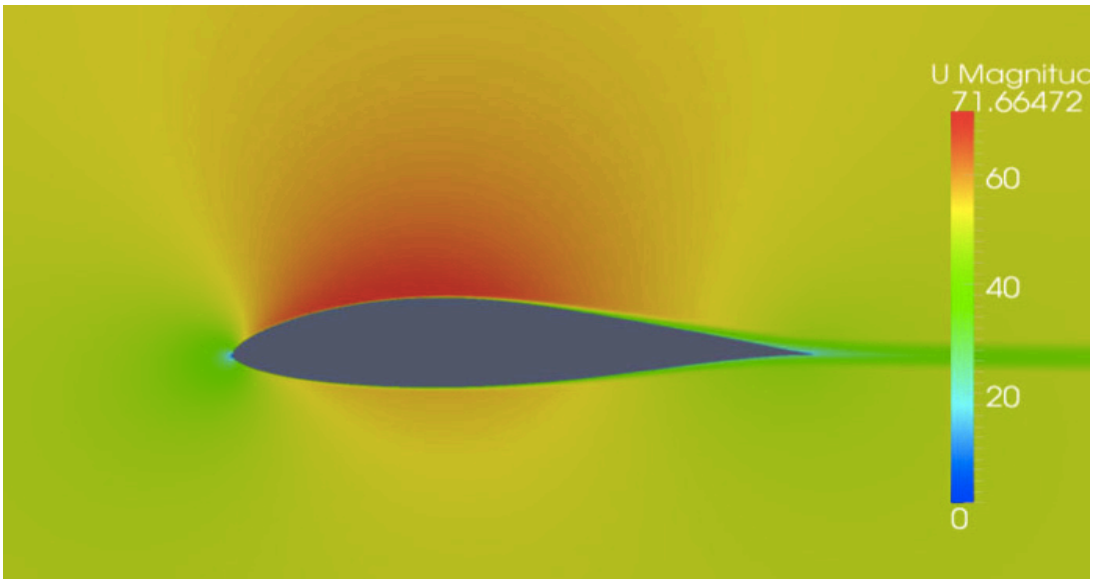

Figure 1: Flow field around an airfoil from a wind turbine, computed with OpenFOAM®. Courtesy of Richard Dwight, aerodynamics group from the Aerospace Engineering Faculty of Delft University of Technology.

OpenFOAM®

OpenFOAM®: OpenFOAM® (www.openfoam.com) is a very popular open source CFD (Computational Fluid Dynamics) package. The fact that OpenFOAM® is free (as opposed to other CFD packages that are usually rather expensive) certainly helps to explain its popularity. But the openness is probably just as important for users. By accessing the source code, they can inspect the methods that have been implemented and extend or improve them according to their needs. This has made it the platform of choice for much of the CFD research that is going on today.

Experiment: Kalman filtering for OpenFOAM®

OpenFOAM® does not yet have a standard facility for data assimilation and calibration. OpenDA might be a good candidate to fill this gap. If an effective combination can be made of OpenDA and OpenFOAM®, it will open up a load of useful functionality for OpenFOAM® users. The OpenDA Association decided to develop a link between OpenFOAM® and OpenDA in close cooperation with the aerodynamics group from the Aerospace Engineering Faculty of Delft University of Technology.

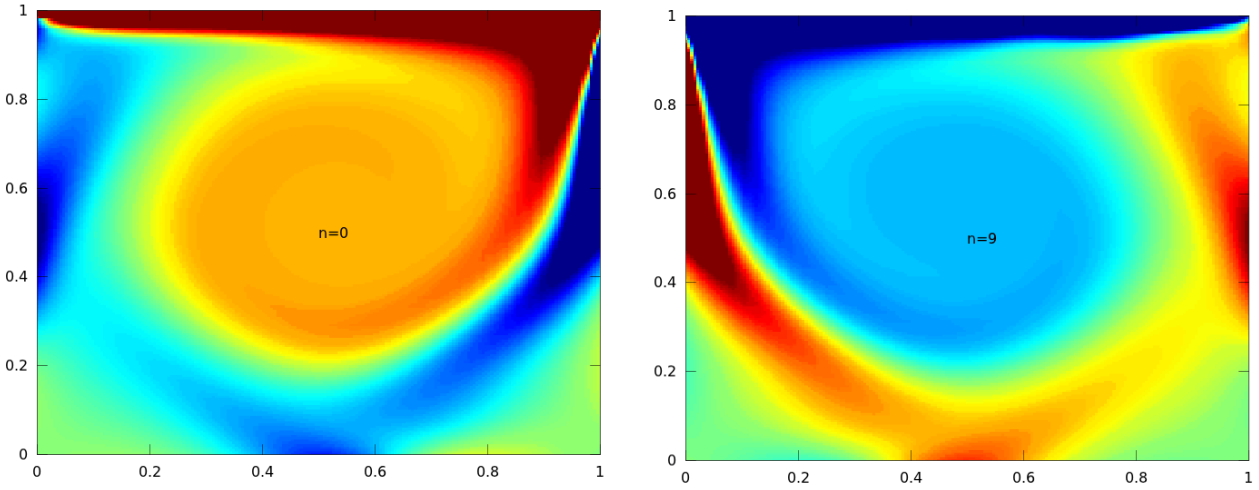

Figure 2: Left: the initial flow field with flow at the lid going to the right. Right: after assimilating observations from a model with leftward flow at the lid the flow in the entire cavity changes its direction.

The connection was done in the simplest way: by using the OpenDA black box wrapper. In this case, OpenDA just reads output files from an OpenFOAM® run and produces modified input files for the next OpenFOAM® timesteps. Although this may seem inefficient, the actual overhead from reading and writing files was only 10% of the total computation time for a significant case.

The OpenDA noise model was extended such that it can now define noise models based on the OpenFOAM® mesh that is used in the computation. Several useful noise models for OpenFOAM® meshes have been implemented. In addition, several performance improvements were introduced in OpenDA to facilitate the handling of really large grids.

Results

Several experiments have been done with the OpenDA/ OpenFOAM® combination. One interesting example is a lid driven cavity, where the OpenFOAM® model initially assumes that the lid induces a flow to the right. Then observations from the same model with the flow at the lid going to the left are fed into the model using OpenDA. Filtering with the OpenDA EnKF filter successfully modifies the flow to be consistent with the observations, flowing left at the lid.

Conclusions

Conclusions: A generic coupling between OpenDA and OpenFOAM® has been implemented, providing a powerful and versatile data assimilation facility for OpenFOAM®.

References

www.openda.org, www.openfoam.com

OPENFOAM® is a registered trademark of SGI Corp