Documentation for Verdandi 1.7

# Contents

## I Introduction to Data Assimilation and Verdandi

### 1 Data Assimilation Principles

1.1 Motivations and basic definitions ........................................... 3

1.2 Fundamental principles ......................................................... 4

1.2.1 Variational procedures ...................................................... 4

1.2.2 Sequential procedures ....................................................... 6

1.2.3 Reduced-order sequential strategies ...................................... 7

1.2.4 Luenberger observers ........................................................ 9

1.2.5 Joint state-parameter estimation with Luenberger observers ......... 9

### 2 Verdandi

2.1 Scientific context ............................................................... 11

2.2 What is Verdandi for? ........................................................... 11

2.3 Acknowledgment ................................................................. 12

## II User's Guide

### 3 Getting Started

3.1 How to Use Verdandi ............................................................ 15

3.1.1 Installation ................................................................. 15

3.1.2 Notation ................................................................. 15

3.1.3 Picking a data assimilation method .................................... 15

3.1.4 Applying the assimilation method to one of the examples ......... 15

3.1.5 Plugging your own model and observations ......................... 15

### 4 Installation

4.1 Dependencies ................................................................. 17

4.2 Linux ................................................................. 17

4.2.1 Installation ................................................................. 17

4.2.2 Installation of dependencies .............................................. 18

4.2.3 Testing the installation .................................................... 18
## Contents

10.3.1 Options ................................................................. 35  
10.3.2 Exceptions raised ..................................................... 37  
10.3.3 Exceptions and debugging .......................................... 37  
10.4 Logger ................................................................. 37  
  10.4.1 Logger status ...................................................... 37  
  10.4.2 Logger file name .................................................... 37  
  10.4.3 Two methods ........................................................ 38  
  10.4.4 Logger options ...................................................... 38  
  10.4.5 Formatting message ............................................... 38  
  10.4.6 Level of priority ................................................... 38  
  10.4.7 Example .......................................................... 38  

11 Python Interface .......................................................... 41  

III Data Assimilation Methods ................................................. 45  

12 Assimilation Methods ...................................................... 47  

13 Ensemble Kalman Filter .................................................. 49  
  13.1 Ensemble Kalman Filter algorithm ................................ 49  

14 Extended Kalman Filter .................................................... 51  
  14.1 Extended Kalman filter algorithm ................................ 51  

15 Four Dimensional Variational ............................................ 53  
  15.1 Four Dimensional Variational algorithm ......................... 54  
    15.1.1 Gradient-based optimization ................................ 54  
    15.1.2 Derivative-free optimization ................................ 54  
    15.1.3 Notation .......................................................... 54  

16 Monte Carlo ............................................................. 55  
  16.1 Monte Carlo algorithm ............................................. 55  

17 Optimal Interpolation .................................................... 57  
  17.1 Optimal interpolation algorithm .................................. 57  

18 Reduced Minimax Filter ................................................ 59  
  18.1 Reduced Minimax filter algorithm ................................. 59  

19 Reduced Order Extended Kalman Filter ............................... 61  
  19.1 Reduced Order Extended Kalman filter algorithm ............... 62  

20 Reduced Order Unscented Kalman Filter ............................... 63  
  20.1 Reduced Order Unscented Kalman filter algorithm ............... 64
# CONTENTS

## 20.2 Simplex case

## 20.3 Generalized case

## 20.4 Reference

## 21 Unscented Kalman Filter

## 21.1 Unscented Kalman filter algorithm

## 21.2 Reference

## IV Parallelism

## 22 Parallelism in Verdandi

## 22.1 Parallel Method applied to Sequential Model

## 22.1.1 Parallel Algorithms

## 22.1.2 Example Programs

## 22.1.3 Performance

## 22.2 Sequential Method applied to Parallel Model

## 22.2.1 Types Management in Verdandi

## 22.2.2 Distributed Structure in Seldon

## 22.2.3 'PETScClampedBar' Distributed Model

## 22.2.4 'ReducedOrderUnscentedKalmanFilter'

## 22.2.5 Example Programs

## 22.3 Parallel Method applied to Parallel Model

## 22.3.1 Parallel 'ReducedOrderUnscentedKalmanFilter'

## 22.3.2 Example Programs

## 22.4 Performance

## V Example Models

## 23 Example Models

## 24 Clamped Bar

## 24.1 The Clamped Bar model

## 24.2 Simulation

## 25 Lorenz model

## 26 Quadratic Model

## 27 Shallow-water

---

Generated on Mon Sep 28 2015 16:49:58 for Verdandi by Doxygen
## CONTENTS

39.1 Configuration options .................................................. 135

### 40 Optimization Solver .................................................. 137

40.1 NLopt Solver ............................................................... 137
   40.1.1 NLopt Installation ................................................. 137
   40.1.2 Using the Class NLoptSolver ................................... 137
   40.1.3 Configuration ...................................................... 137

### IX Developing in Verdandi .............................................. 139

### 41 Dependencies ............................................................ 141

### 42 Linear Algebra Library ............................................... 143
   42.1 Seldon Main Strengths .............................................. 143
   42.2 Seldon Matrices and Vectors ..................................... 144

### 43 Coding Standard ........................................................ 145
   43.1 General Conventions ............................................... 145
      43.1.1 Development .................................................. 145
      43.1.2 Names .......................................................... 145
      43.1.3 Formatting .................................................... 146
      43.1.4 Language Features .......................................... 148
   43.2 C++ ................................................................. 148
      43.2.1 Development .................................................. 148
      43.2.2 Names .......................................................... 148
      43.2.3 Formatting .................................................... 150
      43.2.4 Files ........................................................... 155
      43.2.5 About C++ Features ......................................... 156
      43.2.6 Other Conventions .......................................... 157
   43.3 Python ............................................................... 157
   43.4 Other Languages .................................................... 157

### X Reference Documentation ............................................ 159

### 44 Namespace Index ....................................................... 161
   44.1 Namespace List ..................................................... 161

### 45 Hierarchical Index ..................................................... 163
   45.1 Class Hierarchy ..................................................... 163

### 46 Class Index ............................................................. 165
   46.1 Class List .......................................................... 165
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>48.10.2</td>
<td>Constructor &amp; Destructor Documentation</td>
<td>231</td>
</tr>
<tr>
<td>48.10.3</td>
<td>Member Function Documentation</td>
<td>232</td>
</tr>
<tr>
<td>48.11</td>
<td>Verdandi::ErrorArgument Class Reference</td>
<td>233</td>
</tr>
<tr>
<td>48.11.1</td>
<td>Detailed Description</td>
<td>233</td>
</tr>
<tr>
<td>48.11.2</td>
<td>Constructor &amp; Destructor Documentation</td>
<td>234</td>
</tr>
<tr>
<td>48.11.3</td>
<td>Member Function Documentation</td>
<td>234</td>
</tr>
<tr>
<td>48.12</td>
<td>Verdandi::ErrorConfiguration Class Reference</td>
<td>234</td>
</tr>
<tr>
<td>48.12.1</td>
<td>Detailed Description</td>
<td>235</td>
</tr>
<tr>
<td>48.12.2</td>
<td>Constructor &amp; Destructor Documentation</td>
<td>235</td>
</tr>
<tr>
<td>48.12.3</td>
<td>Member Function Documentation</td>
<td>235</td>
</tr>
<tr>
<td>48.13</td>
<td>Verdandi::ErrorIO Class Reference</td>
<td>236</td>
</tr>
<tr>
<td>48.13.1</td>
<td>Detailed Description</td>
<td>237</td>
</tr>
<tr>
<td>48.13.2</td>
<td>Constructor &amp; Destructor Documentation</td>
<td>237</td>
</tr>
<tr>
<td>48.13.3</td>
<td>Member Function Documentation</td>
<td>237</td>
</tr>
<tr>
<td>48.14</td>
<td>Verdandi::ErrorProcessing Class Reference</td>
<td>238</td>
</tr>
<tr>
<td>48.14.1</td>
<td>Detailed Description</td>
<td>238</td>
</tr>
<tr>
<td>48.14.2</td>
<td>Constructor &amp; Destructor Documentation</td>
<td>238</td>
</tr>
<tr>
<td>48.14.3</td>
<td>Member Function Documentation</td>
<td>239</td>
</tr>
<tr>
<td>48.15</td>
<td>Verdandi::ErrorPythonUndefined Class Reference</td>
<td>239</td>
</tr>
<tr>
<td>48.15.1</td>
<td>Detailed Description</td>
<td>240</td>
</tr>
<tr>
<td>48.15.2</td>
<td>Constructor &amp; Destructor Documentation</td>
<td>240</td>
</tr>
<tr>
<td>48.15.3</td>
<td>Member Function Documentation</td>
<td>241</td>
</tr>
<tr>
<td>48.16</td>
<td>Verdandi::ErrorUndefined Class Reference</td>
<td>241</td>
</tr>
<tr>
<td>48.16.1</td>
<td>Detailed Description</td>
<td>242</td>
</tr>
<tr>
<td>48.16.2</td>
<td>Constructor &amp; Destructor Documentation</td>
<td>242</td>
</tr>
<tr>
<td>48.16.3</td>
<td>Member Function Documentation</td>
<td>242</td>
</tr>
<tr>
<td>48.17</td>
<td>Verdandi::ExtendedKalmanFilter Class Reference</td>
<td>243</td>
</tr>
<tr>
<td>48.17.1</td>
<td>Detailed Description</td>
<td>245</td>
</tr>
<tr>
<td>48.17.2</td>
<td>Constructor &amp; Destructor Documentation</td>
<td>245</td>
</tr>
<tr>
<td>48.17.3</td>
<td>Member Function Documentation</td>
<td>246</td>
</tr>
<tr>
<td>48.18</td>
<td>Verdandi::ForwardDriver Class Reference</td>
<td>247</td>
</tr>
<tr>
<td>48.18.1</td>
<td>Detailed Description</td>
<td>249</td>
</tr>
<tr>
<td>48.18.2</td>
<td>Constructor &amp; Destructor Documentation</td>
<td>249</td>
</tr>
<tr>
<td>48.18.3</td>
<td>Member Function Documentation</td>
<td>249</td>
</tr>
<tr>
<td>48.19</td>
<td>Verdandi::FourDimensionalVariational Class Reference</td>
<td>250</td>
</tr>
<tr>
<td>48.19.1</td>
<td>Detailed Description</td>
<td>252</td>
</tr>
<tr>
<td>48.19.2</td>
<td>Constructor &amp; Destructor Documentation</td>
<td>253</td>
</tr>
<tr>
<td>48.19.3</td>
<td>Member Function Documentation</td>
<td>253</td>
</tr>
<tr>
<td>48.20</td>
<td>Verdandi::GridToNetworkObservationManager Class Reference</td>
<td>254</td>
</tr>
<tr>
<td>48.20.1</td>
<td>Detailed Description</td>
<td>260</td>
</tr>
</tbody>
</table>
48.41.1 Detailed Description ........................................ 429
48.41.2 Constructor & Destructor Documentation ................. 429
48.41.3 Member Function Documentation ............................ 429
48.42 Verdandi::QuadraticModel Class Reference .................. 432
48.42.1 Detailed Description ....................................... 436
48.42.2 Constructor & Destructor Documentation .................. 436
48.42.3 Member Function Documentation ............................ 437
48.43 QuadraticModel::QuadraticModel Class Reference ........... 444
48.43.1 Detailed Description ....................................... 445
48.43.2 Constructor & Destructor Documentation .................. 445
48.43.3 Member Function Documentation ............................ 446
48.44 Verdandi::ReducedMinimax Class Reference .................. 450
48.44.1 Detailed Description ....................................... 453
48.44.2 Constructor & Destructor Documentation .................. 454
48.44.3 Member Function Documentation ............................ 454
48.45 Verdandi::ReducedOrderExtendedKalmanFilter Class Reference ............................... 456
48.45.1 Detailed Description ....................................... 458
48.45.2 Constructor & Destructor Documentation .................. 458
48.45.3 Member Function Documentation ............................ 459
48.46 Verdandi::ReducedOrderUnscentedKalmanFilter Class Reference .................. 460
48.46.1 Detailed Description ....................................... 463
48.46.2 Constructor & Destructor Documentation .................. 463
48.46.3 Member Function Documentation ............................ 463
48.47 Verdandi::ShallowWater Class Reference ...................... 465
48.47.1 Detailed Description ....................................... 470
48.47.2 Constructor & Destructor Documentation .................. 470
48.47.3 Member Function Documentation ............................ 471
48.48 Verdandi::TR1PerturbationManager Class Reference ........... 477
48.48.1 Detailed Description ....................................... 478
48.48.2 Constructor & Destructor Documentation .................. 479
48.48.3 Member Function Documentation ............................ 479
48.49 Verdandi::TrajectoryManager Class Reference ............... 482
48.49.1 Detailed Description ....................................... 483
48.49.2 Member Function Documentation ............................ 483
48.50 Verdandi::TRNGPerturbationManager Class Reference ......... 484
48.50.1 Detailed Description ....................................... 485
48.50.2 Constructor & Destructor Documentation .................. 486
48.50.3 Member Function Documentation ............................ 486
48.51 Verdandi::UnscentedKalmanFilter Class Reference ............ 488
48.51.1 Detailed Description ....................................... 491
Part I

Introduction to Data Assimilation and Verdandi
Chapter 1

Data Assimilation Principles

1.1 Motivations and basic definitions

In order to obtain some information – as detailed as possible – on a natural system, such as in geophysics, or regarding the important subcategory of living systems which constitute the objects of study in biology and medicine, the most straightforward strategy consists in obtaining measures on the system at hand. Note that we deliberately use the term measurement, related but not reduced to experiments, to signify that it is not in general possible to design specific experiments allowing to determine all kinds of physical properties of the system – as is done for instance for industrial systems. By contrast, natural systems induce drastic limitations in measurements, in that they generally must be “taken as they are”, namely, observed in their current operating conditions, whether this is due to the practical impossibility of apprehending them comprehensively (e.g. in geophysics), or to the undesirable character of any strong perturbation of the system (invasiveness in living systems).

Specializing now our discussion on living systems, abundant measurements are frequently at hand, in the form of clinical images and signals of various origins. Despite the diversity and rich information contents of these data, they are also inevitably limited in many respects. Beside considerations on sampling and noise, this holds in particular as regards:

- their extent: for example only 2D measurements, or boundary information, may be available for a 3D system, or only part of the whole domain;
- their type: some quantities are never measured, such as internal stresses in a living tissue, or various physical constitutive parameters (stiffness, contractility, etc.).

Nevertheless, we may want to consider physical models to describe and predict the behavior of these systems. Clearly, these models may be seen as providing some complementary information on the system. However, their predictivity requires the careful adjustment of many parameters – in particular regarding the detailed geometry (anatomy), physical properties, boundary conditions and initial conditions needed in the model – most of which being out of reach of the available measurements.

The purpose of data assimilation is then to combine the information available from these two sources – measurements on the one hand, and models and the other hand – by seeking an adequate compromise between:

- the discrepancy computed between simulations of the model and the corresponding measurements;
- the a priori confidence in the model, since errors are also present in the measurements.

The desired output of this procedure is an estimation of the unknown quantities of interest, namely,

- state variables (the “trajectories” of the system), and in particular their initial values at a given reference time;
- physical parameters which must be prescribed in the model equations.

In terms of clinical applications, the expected benefits are to assist and improve both diagnosis and prognosis:
Data Assimilation Principles

– diagnosis, by providing more complete information on the patient (spatially-distributed quantities, and various otherwise unreachable indicators), and with improved accuracy;

– prognosis, since once data assimilation has been performed the model can be more confidently used to predict natural or artificial evolutions of the system, for example to simulate the effect of various possible therapeutic strategies.

We will now introduce some basic notation necessary to discuss the fundamental principles of data assimilation. First of all, we consider physical models in the form of dynamical systems governed by equations of the type

\[ \dot{x} = M(x, p, t). \] (1.1)

In this equation, \( x \) denotes the so-called state variable, namely, the physical quantity which the model aims at describing in its time-wise evolution – hence, the time derivative in the left-hand side – and also frequently spatial variations for distributed quantities. In this generic notation, the whole model is essentially summarized in the so-called dynamical operator \( M \), which applies on the state variable itself, and may depend on time \( t \) as well as on a set of physical parameters denoted by \( p \). This operator may arise from various types of physical formulations, e.g. in solid and fluid mechanics or electrophysiology. Mathematically speaking, it may take the form of partial differential equations (PDEs) or ordinary differential equations (ODEs, namely, only differentiated with respect to the time variable), or algebraic systems, in particular.

Clearly, in such model formulations we need to prescribe – hence to estimate when unknown via the data assimilation procedure – the initial condition \( M(0) \) and the parameter vector \( p \). Typically, in the models considered the state variable may contain a large number of scalar coefficients – typically \( 10^3 \) to \( 10^7 \) degrees of freedom in a continuum mechanics model – whereas the size of the parameter vector is generally much more limited, and in practice we seldom have to estimate more than a few hundreds of parameter values. Once the initial condition and the parameter vector associated with (1.1) are estimated, the model can be simulated in time, using appropriate numerical techniques.

Another important notation concerns the measurements, typically represented by an equation of the type

\[ y = H(x, t) + e, \] (1.2)

where \( y \) denotes the actual data, \( H \) is the so-called observation operator, and \( e \) accounts for the error inherent to the measurement process, often called the noise. Note that the quantity \( y \) will frequently correspond to pre-processed – not raw – data, for example images processed with segmentation or optical flow techniques in order to extract some position, displacement or velocity information. We further emphasize that this equation also represents a model – in this case of the measurements – where modeling ingredients are embedded both in the expression of \( H \) and the characterization of \( e \), which may be of probabilistic or deterministic nature.

1.2 Fundamental principles

We are now in a position to introduce some fundamental principles for data assimilation procedures. First of all, there are two main categories of methods: variational and sequential methods.

1.2.1 Variational procedures

In variational procedures, we consider a criterion to be minimized in order to achieve the above-mentioned compromise between simulation-measurements discrepancy and model confidence, see e.g. [2, 3] and references therein.

A typical criterion would read

\[ J_T(\xi_x, \xi_\theta) = \int_0^T \|y - H(x)\|^2_M dt + \|\xi_x\|_{(R_x)}^{-1} + \|\xi_\theta\|_{(R_\theta)}^{-1}, \] (1.3)

where \( \| \cdot \|_M, \| \cdot \|_{(R_x)}^{-1} \) and \( \| \cdot \|_{(R_\theta)}^{-1} \) denote suitable norms for each quantity concerned, and associated with the matrices appearing as subscripts. In this criterion, \( x \) is constrained to satisfy the model equation (1.1) starting from the initial condition \( x(0) = x_0 + \xi_x \) and with parameter values given by \( \theta = \theta_0 + \xi_\theta \). In order to perform this minimization, a classical strategy consists in computing the gradient of the criterion, which requires the simulation
of the so-called adjoint model. The adjoint model equation is an evolution system closely related to – and inferred from, indeed – the direct model equation (1.1), viz.

\[
\begin{aligned}
\dot{u}_x + \frac{\partial \mathcal{H}}{\partial x}^T u_x &= - \frac{\partial \mathcal{H}}{\partial x}^T M(y - \mathcal{H}(x)) \\
u_x(T) &= 0 \\
\dot{u}_\theta + \frac{\partial \mathcal{H}}{\partial \theta}^T u_x &= 0 \\
u_\theta(T) &= 0
\end{aligned}
\] (1.4)

These equations must be simulated backwards in time from the final time \(T\) to the initial time in order to obtain the gradient value criterion expressed as

\[
\begin{aligned}
d_{\xi_x} &\mathcal{J} \cdot \delta \xi_x = \xi_x^T (P_0)^{-1} \delta \xi_x - u_x(0)^T \delta \xi_x, \\
d_{\xi_\theta} &\mathcal{J} \cdot \delta \theta = \xi_\theta^T (P_*)^{-1} \delta \xi_\theta - u_\theta(0)^T \delta \xi_\theta,
\end{aligned}
\]

Hence, each gradient computation requires the forward simulation of the direct model and the backward simulation of the adjoint, and this must be repeated until convergence of the minimization algorithm. Figure 1.1 illustrates this iterative procedure, and the fact that observability can be improved by considering a longer time window, hence, also more measurements. This type of variational procedure is also referred to as “4DVar” in the data assimilation community, while “3DVar” is used to refer to minimization estimation performed for static models, or for dynamic models at a given time (namely, without time integral).

**Figure 1.1:** Schematic of minimization iterations and Kalman filter tracking on a sliding time window (as seen via one single output)

Concerning the time discretization, it is classical to formulate an optimal discrete time minimization criterion and find its corresponding adjoint rather than discretizing directly (1.4). Hence, we consider a criterion of the form

\[
J_N(\xi_x, \xi_\theta) = \sum_{k=0}^{N} \|y_k - \mathcal{H}(x_k)\|_{M_k}^2 + \|\xi_x\|_{(P_0)^{-1}}^2 + \|\xi_\theta\|_{(P_*)^{-1}}^2,
\] (1.5)

with for example \(M_k = \Delta t M\).
1.2.2 Sequential procedures

By contrast, sequential procedures – also often referred to as filtering – proceed by simulating equations closely resembling the direct model equations, with an additional correction term taking into account the discrepancy between the simulation and the actual measurements, namely \( y - \mathcal{H}(x) \), quantity called the innovation. For example when only the initial condition is unknown the filtering equation would be of the type

\[
\dot{x}^a = \mathcal{H}(x^a, t) + K(y - \mathcal{H}(x^a)),
\]

(1.6)

where the operator \( K \) – frequently linear – is called the filter. The filtering equation simulation is then started from the candidate initial condition \( x_0 \), and the aim of the correction is to bring the simulated trajectory close to the target system, see Fig. 1.1.

This type of strategy was made extremely popular by the Kalman theory, which formulated an optimal setting for deriving the filter operator, initially when the operators \( A \) and \( H \) are both linear. In this case, the Kalman equations read

\[
\begin{align*}
\dot{x}^a &= \mathcal{H}(x^a) + P_0 \mathcal{H}^T M (y - \mathcal{H}(x^a)) \\
P &= P_0 - P_0 \mathcal{H}^T P + P \mathcal{H}^T M \mathcal{H} P = 0 \\
P(0) &= P_0 \\
x^a(0) &= x_0
\end{align*}
\]

(1.7)

where \( W \) and \( P_0 \) denote the so-called covariance matrices of the measurement noise and initial condition uncertainty, respectively. We can see that the filter expression is based on the computation of the time-dependent covariance matrix \( P \) which satisfies a Riccatti equation. In fact, in the linear case the variational and Kalman procedures can be shown to be exactly equivalent, and the minimizing direct and adjoint states (\( X_{\text{inf}} \) and \( p_{\text{inf}} \), respectively) are related to the Kalman filter by the identity [2]

\[
x_{\text{inf}} = x^a + P x_{\text{inf}}.
\]

This is also illustrated in Figure 1.1 where we show that the Kalman estimation coincides with the minimizing solution at the end of the time window considered.

When nonlinearities are to be considered, various extensions are available, and in particular the Extended Kalman Filtering (EKF) approach, in which the linearized forms of the operators are used in the filter equation. However, in such a case the approach is no longer equivalent to the variational setting. Nevertheless, some alternative filter equations can be derived from the variational formulation, but the filter computation then requires solving a Hamilton-Jacobi-Bellman equation in a space which has the dimension of the state variable [8], which is in general not practical.

Note that, in practice, the Kalman (or EKF) approach itself is also quite limited as regards the size of the system which can be handled, since the covariance matrix \( P \) has the size of the state variable, and is a dense matrix, unlike the dynamical operators. In order to circumvent this limitation some alternative approaches must be considered.

Concerning time discretization, it is classical to formulate the optimal filter directly from the optimal time-discrete minimization criteria rather than discretizing directly (1.7). Hence, after some quite tedious computations, we can formulate a prediction-correction scheme

1. Prediction:

\[
\begin{array}{l}
\dot{\xi}^{f}_{h+1} = \mathcal{H}_h(\xi^a_{h+1}) \\
P^{f}_{h+1} = \frac{\partial \mathcal{H}_h}{\partial x} P^{a}_{h+1} \frac{\partial \mathcal{H}_h}{\partial x}^T
\end{array}
\]

(1.8a)

2. Correction:

\[
\begin{array}{l}
P^{a}_{h+1} = \left( \frac{\partial \mathcal{H}_h}{\partial x} R_{h+1}^{-1} \frac{\partial \mathcal{H}_h}{\partial x}^T + (P^{f}_{h+1})^{-1} \right)^{-1} \\
G_{h+1} = P^{a}_{h+1} \frac{\partial \mathcal{H}_h}{\partial x} R_{h+1}^{-1} \\
\xi^a_{h+1} = \xi^a_{h} + G_{h+1} (y_{h+1} - \mathcal{H}_h(\xi^a_{h} + P^{f}_{h+1} G_{h+1}^{-1} (y_{h+1} - \mathcal{H}_h(\xi^a_{h})))))
\end{array}
\]

(1.8b)

For the sake of simplicity, we have introduced the Kalman filter in a deterministic context, but the probabilistic counterpart exists. In a linear framework the expressions are exactly the same, albeit with the additional interpretation
1.2 Fundamental principles

\[ \begin{align*}
\text{a priori mean: } \hat{x}_{h+1}^f &= \mathbb{E}(x_{h+1}|y_0, \ldots, y_h), \\
\text{a priori covariance: } P_{h+1}^f &= \mathbb{E}((x_{h+1} - \hat{x}_{h+1}^f)(x_{h+1} - \hat{x}_{h+1}^f)^T), \\
\text{a posteriori mean: } \hat{x}_{h+1} &= \mathbb{E}(x_{h+1}|y_0, \ldots, y_{h+1}), \\
\text{a posteriori covariance: } P_{h+1} &= \mathbb{E}((x_{h+1} - \hat{x}_{h+1})(x_{h+1} - \hat{x}_{h+1})^T).
\end{align*} \]  

(1.9)

This results extend to the non-linear context with the EKF (1.8) but the identities (1.9) are then only approximate. To improve the quality of this approximation, the Unscented Kalman Filter has then been introduced [6], based on the idea of substituting means and covariances by empirical quantities computed from sample points:

\[ \begin{align*}
\hat{x}_{h+1}^c &= \sum_{i=1}^d \alpha_i x_{h+1}^{i[1]}, \\
\hat{P}_{h+1}^c &= \sum_{i=1}^d \alpha_i (x_{h+1}^{i[1]} - \hat{x}_{h+1}^c)(x_{h+1}^{i[1]} - \hat{x}_{h+1}^c)^T, \\
\hat{x}_{h+1} &= \sum_{i=1}^d \alpha_i (x_{h+1}^{i[1]} - \hat{x}_{h+1}^c)(x_{h+1}^{i[1]} - \hat{x}_{h+1})^T, \\
\hat{P}_{h+1} &= \sum_{i=1}^d \alpha_i (x_{h+1}^{i[1]} - \hat{x}_{h+1})^T (x_{h+1}^{i[1]} - \hat{x}_{h+1}).
\end{align*} \]  

(1.10)

with \( \sum_{i=1}^d \alpha_i = 1 \).

In practice, the correction particles are sampled around the mean \( \hat{x}_{h+1}^c \) with a covariance \( P_{h+1}^c \) and the prediction samples then verify

\[ \hat{x}_{h+1} = \mathcal{H}(\hat{x}_{h+1}^c). \]  

(1.11a)

Then, by computing

\[ y_{h+1} = \mathcal{H}(\hat{x}_{h+1}|y_h) = \sum_{i=1}^d \alpha_i y_{h+1}^{i[1]}, \quad \hat{y}_{h+1} = \sum_{i=1}^d \alpha_i y_{h+1}^{i[1]} \]  

(1.11b)

The gain is defined by

\[ \begin{align*}
G_{h+1} &= (\hat{P}_{h+1}^c)^{-1} \cdot (\hat{P}_{h+1}^{xy})^{-1}, \\
\hat{P}_{h+1}^{xy} &= \sum_{i=1}^d \alpha_i (x_{h+1}^{i[1]} - \hat{x}_{h+1}^c)(y_{h+1}^{i[1]} - \hat{y}_{h+1})^T, \\
\hat{P}_{h+1}^{xy} &= \sum_{i=1}^d \alpha_i (y_{h+1}^{i[1]} - \hat{y}_{h+1})(y_{h+1}^{i[1]} - \hat{y}_{h+1})^T + W_{h+1}
\end{align*} \]  

(1.11c)

so that we keep having

\[ \begin{align*}
\hat{x}_{h+1} &= \hat{x}_{h+1}^c + G_{h+1}(y_{h+1} - \hat{y}_{h+1}), \\
P_{h+1} &= \hat{P}_{h+1}^c - P_{h+1}^{xy} \cdot (\hat{P}_{h+1}^{xy})^{-1} \cdot (\hat{P}_{h+1}^{xy})^T
\end{align*} \]  

(1.11d)

and proceed recursively with new correction particles \( \hat{x}_{h+1}^{i[1]} \).

The Ensemble Kalman Filter, introduced in [4], follows the same principles of approximating the covariances by sampled particles with, most of the time, an increased number of particles with respect to the UKF filter, and various ways of sampling the particles around the mean value. Finally Monte Carlo strategies exploit a very large number of particles to give a better approximation of the non-linear optimal filter but the practical details of these methods are beyond the scope of this review focused on large dimensional systems coming from the discretization of PDEs.

1.2.3 Reduced-order sequential strategies

1.2.3.0.1 Reduced-Order Extended Kalman Filtering (ROEKF)

In order to deal with the limitations of sequential strategies due to the system size, a classical strategy consists in assuming a specific reduced-order form for the covariance operators. For example, making the ansatz

\[ \forall t, \quad P(t) = L(t) U(t)^{-1} L(t)^T \]  

(1.12)

with \( U \) an invertible matrix of small size \( r \) and \( L \) an extension operator, we can show that within linear assumptions the solution of the Riccatti equation in (1.7) reduces to

\[ \dot{L} = \mathcal{H} L \text{ and } \dot{U} = L U M \mathcal{H} L. \]

(1.13)

which is actually computable in practice.
In a non-linear framework, we can then approximate the covariance dynamics by extending (1.13) as

\[ L = \frac{\partial \mathcal{M}}{\partial x} L \text{ and } U = L^T \frac{\partial \mathcal{H}^T}{\partial x} M \frac{\partial \mathcal{H}}{\partial x} L. \]  

(1.14)

These strategies have relevant applications in the case of parameter identification per se. It is common, indeed, to assume more space regularity for the parameters than for the state initial condition. Hence, after discretization the parameters can be represented by a small number of degrees of freedom. Assuming that we can limit \( U(0) \) to the parametric space, the extension operator can then be decomposed into two components \( L = (L_x \ | \ L_\theta) \) with \( L_\theta = \delta \xi \) and

\[ L_x = \frac{\partial \mathcal{M}}{\partial x} L + \frac{\partial \mathcal{M}}{\partial \theta}. \]  

(1.15)

We recognize in this expression the dynamics of the sensitivity operator \( \frac{\partial \mathcal{H}}{\partial \theta} \), which provides a nice interpretation for this strategy of uncertainty covariance reduction. Furthermore, in the linear framework we can prove that this sequential estimator corresponds to the optimal filter associated with the criterion

\[ \mathcal{J}(\xi_\theta) = \int_0^T \| y - \mathcal{H}(x) \|_2^2 dt + \| \xi_\theta \|_2^2 (P_1)^{-1}, \]  

(1.16)

where \( x \) follows the trajectory associated with \( \xi_\theta \) and fixed initial condition, which is commonly used in variational identification procedures. All this justifies naming this strategy Reduced-Order Extended Kalman Filter (ROEKF), but it is also known as Singular Evolutive Extended Kalman Filter following the work of [15].

This concept can be applied directly on time and space discretized versions of the equations, which leads to a discrete time Reduced-Order Extended Kalman Filter.

1. Prediction:

\[
\begin{align*}
\xi_{h+1}^f &= \mathcal{H}(x_h^d) \\
L_{h+1} &= \frac{\partial \mathcal{H}}{\partial x} L_h
\end{align*}
\]  

(1.17a)

2. Correction:

\[
\begin{align*}
U_{h+1} &= U_h + L_{h+1}^T \frac{\partial \mathcal{H}}{\partial x} L_{h+1} R_{h+1}^{-1} \frac{\partial \mathcal{H}}{\partial x} L_h \\
G_{h+1} &= P_{h+1}^\alpha \frac{\partial \mathcal{H}}{\partial x} L_{h+1} R_{h+1}^{-1} \\
\tilde{x}_{h+1}^\alpha &= \tilde{x}_{h+1}^f + G_{h+1}(y_{h+1} - \frac{\partial \mathcal{H}}{\partial x} \tilde{x}_{h+1}^f)
\end{align*}
\]  

(1.17b)

1.2.3.0.2 Reduced-Order Unscented Kalman Filtering (ROUKF)

Alternatively, this strategy can be coupled with the UKF approach by showing that particles can be generated only in the space of small dimension and the computation made in the UKF filter (1.11) can be compatible with the time discretized counterpart of (1.12). This was proven in [9, 10] which also provided a very general version of the Reduced Order Unscented Kalman Filter (ROUKF). In particular this algorithm is close to the Singular Evolutive Interpolated Kalman Filter [14, 5] for a choice of particles \( d = r + 1 \) and reads as follows.

1.2.3.0.3 Algorithm –

Given an adequate sampling rule, we store the corresponding weights in the diagonal matrix \( D_\alpha \) and precompute so-called unitary sigma-points (i.e. with zero mean and unit covariance) denoted by \( (\tilde{\xi}_i^d)_{1 \leq i \leq r+1} \); we then perform at each time step

1. Sampling:

\[
\begin{align*}
C_h &= \sqrt{(U_h)^{-1}} \\
\xi_h^{d+i} &= \xi_h^d + L_h \cdot C_h^T \tilde{\xi}_i^d, \quad 1 \leq i \leq r + 1
\end{align*}
\]  

(1.18a)

2. Prediction:

\[
\begin{align*}
\xi_{h+1}^f &= \mathcal{H}(\xi_h^{d+i}), \quad 1 \leq i \leq r + 1 \\
\xi_{h+1}^\alpha &= \sum_{i=1}^{r+1} \alpha_i \xi_{h+1}^{d+i}
\end{align*}
\]  

(1.18b)
### 1.2 Fundamental principles

#### 3. Correction:

\[
\begin{align*}
L_{h+1} &= [\vec{x}_{h+1}]_{a} \mathbf{D}_{a}[\vec{I}]^{T} \\
y_{h+1} &= \mathbf{H}_{h+1} (\hat{x}_{h+1}^{i}) \\
y_{h+1}^{f} &= \sum_{i=1}^{r+1} \alpha_{i} y_{h+1}^{i} \\
\mathbf{U}_{h+1} &= 1 + \hat{x}_{h+1}^{f} R_{h+1}^{-1} \mathbf{U}_{h+1} \\
\hat{x}_{h+1}^{a} &= \hat{x}_{h+1}^{f} - L_{h+1} \mathbf{U}_{h+1} \mathbf{U}_{h+1} \mathbf{U}_{h+1}^{-1} (y_{h+1}^{a} - y_{h+1}^{f})
\end{align*}
\]

(1.18c)

where we denote by \([\vec{I}]^{*}\) the matrix concatenating the \((\vec{I})^{*}\) vectors side by side, and similarly for other vectors.

#### 1.2.4 Luenberger observers

The so-called observer theory, initiated by Luenberger [7], is based on the simple realization that, defining the estimation error

\[
\hat{x} = x - x^{a},
\]

we obtain in the linear case, when subtracting the direct model and filtered equations, the dynamics

\[
\dot{\hat{x}} = (\mathbf{H} - \mathbf{G} \mathbf{H}) \hat{x} - \mathbf{G} \hat{x}.
\]

(1.19)

This type of dynamical equation is well-known in control theory: it is similar to the closed-loop controlled equation of a system of natural dynamics governed by \(\mathbf{H}\), and submitted to a feedback control defined by the operator \(\mathbf{G}\) applied on the quantity observed through \(\mathbf{H}\). Hence, obtaining an accurate estimation of the state variable is exactly equivalent to driving the estimation error \(\hat{x}\) to zero – namely, stabilizing this error – using the feedback control \(\mathbf{G}\).

This approach opened new avenues for formulating novel filtering approaches, because for actual dynamical systems control and stabilization motivations have frequently already led to the formulation of effective feedback controls, used in a large variety of industrial systems. Hence, these approaches can be quite directly adapted to obtain adequate filters which – unlike the Kalman filter – are tractable in practice for large systems. Moreover, these filters are often deeply rooted in the physics of the system considered, hence the computational building blocks needed are likely to be already available in the system simulation software. However, in the Luenberger approach we lose the Kalman optimality, which of course only holds in quite restricted (linear) cases.

For examples of such approaches applicable to biomechanics we refer to [12]. We also point out that the Luenberger observer approach is also sometimes referred to as “nudging” in the data assimilation community [1].

#### 1.2.5 Joint state-parameter estimation with Luenberger observers

As apparent in the above discussion, Luenberger observers were originally designed for state estimation. When parameters are to be jointly estimated, it is quite classical in the filtering context to complement the state equation (1.1) with the artificial parameter dynamics

\[
\dot{\hat{p}} = 0.
\]

(1.20)

Then the whole estimation objective is to estimate the initial condition of the so-called augmented state \((\hat{X}, \hat{p})\). Of course, when a Kalman approach is out of reach for the state variable alone, it holds a fortiori for the augmented state. On the other hand, devising a Luenberger observer for the augmented state is difficult, because part of the dynamics is non-physical, hence feedback controls are not readily available for the joint system.

Nevertheless, an effective approach for joint state-parameter estimation was proposed in [11], based on a Luenberger observer applied on the state equation alone. In essence, this first stage state estimation reduces the uncertainty to the parameter space, which allows to consider a Kalman-like approach (or EKF-like in nonlinear...
cases) for handling the remaining parameter uncertainty. This algorithm can be summarized as

\[
\begin{align*}
\dot{x}^a &= \mathcal{M}(x^a, \hat{\theta}) + G_x(y - \mathcal{H}x^a) + L_x \dot{\hat{\theta}} \\
\hat{\theta} &= U^{-1}L_x^\top \mathcal{H}^\top M(y - \mathcal{H}x^a) \\
L_x &= \left( \frac{\partial \mathcal{M}}{\partial x} - G_x \mathcal{H} \right)L_x + \frac{\partial \mathcal{M}}{\partial \theta} \\
U &= L_x^\top \mathcal{H}^\top M \mathcal{H} L_x \\
x^a(0) &= x_0 \\
\hat{\theta}(0) &= \theta_0 \\
L_x(0) &= 0 \\
U(0) &= U_0
\end{align*}
\]

(1.21)

where \( \mathcal{K}_x \) denotes the state filter (Luenberger observer), \( L_x \) represents the sensitivity of the state variable with respect to the parameters, and \( U \) the inverse of the parameter estimation error covariance. This methodology is strongly related to so-called reduced filtering approaches, see e.g. [15], since essentially only the part of the dynamics concerning parameters is handled using optimal filtering.

This joint estimation approach was later extended in [9] towards strategies inspired from so-called unscented filtering methods – also related to Ensemble Kalman filtering and particle filtering – allowing to avoid the computation of differentiated operators required in (1.21).
Chapter 2

Verdandi

Verdandi is a generic C++ library for data assimilation.

Verdandi is currently developed at INRIA. It aims at providing methods and tools for data assimilation. It is designed to be relevant to a large class of problems involving high-dimensional numerical models.

To guarantee the highest performance, the library is implemented in C++. In addition, Verdandi provides a Python interface generated by Swig.

Models implemented in Fortran, C, C++, Python, ... can be plugged to Verdandi using either a C++ or Python interface.

Verdandi is provided under the GNU Lesser General Public License (LGPL).

2.1 Scientific context

Data assimilation is the process of combining different sources of information in order to better estimate the state of a system. By extension, some parameters can also be estimated. These methods were originally introduced to deal with uncertainties present in models pertaining mostly to geophysics, but it is now widely recognized that they have a tremendous potential in many other applications (see euHeart example below).

Whether the system be biological, environmental, mechanical, etc., the main sources of information are always a numerical model, observations and error statistics. Data assimilation methods can be written independently of the system to which they are applied, and each method can be applied to a wide class of systems. Therefore methods are generic and can be put together in a library.

2.2 What is Verdandi for?

What is Verdandi designed for?

- to provide data assimilation methods to non-specialists;
- to facilitate the application of methods to a great number of problems;
- to provide a framework for perennial development;
- to improve the diffusion and impact of data assimilation algorithms.

Who can be a Verdandi user?

- non-specialists, engineers or researchers, who could directly use the available data assimilation methods;
- a specialist taking advantage of a modular framework, which should ease development, transfers and interactions.
The users provide the numerical model and the observations with the appropriate interface (see the pages models and Observations).

2.3 Acknowledgment

The development of Verdandi is financially supported by the European research initiative euHeart which aims at developing, sharing and integrating patient-specific multi-physics and multi-level models of the heart and great vessels in normal and pathological conditions to address clinical challenges. In this project data assimilation is thus intended to allow the personalization of the biophysical models considered in order to perform genuinely patient-specific simulations.

Since 2011, Verdandi is also partly supported by the European research initiative VPH-Share, specifically as regards high-performance computing features. VPH-Share aims at developing the infostructure and integrating the optimized services to expose and share data and knowledge, jointly developing multiscale models for the composition of new VPH workflows, and facilitating collaborations within the VPH community.

Part II

User’s Guide
Chapter 3

Getting Started

3.1 How to Use Verdandi

3.1.1 Installation

First read the installation section. At the end of the section, you should be able to run an example program.

3.1.2 Notation

Read the notation section so as to be comfortable with the variables used in the documentation.

3.1.3 Picking a data assimilation method

You can browse the available assimilation methods. If you do not know which method to choose, you may try with the simplest method called optimal interpolation.

3.1.4 Applying the assimilation method to one of the examples

You may apply the selected assimilation method to one of the example models. Not all models support all assimilation methods. You can always pick the quadratic model (which is initially configured as a linear model).

You should launch a few simulations with and without assimilation. You may analyze and display the results with the tools provided in Verdandi.

3.1.5 Plugging your own model and observations

Once you applied a data assimilation method and got familiar with Verdandi, you can start writing an interface for your model and an observation manager.
Chapter 4

Installation

Verdandi is supposed to be fully compliant with the C++ standard. This ensures portability on many platforms. It compiles at least on Linux, MacOS and windows. It is compatible with IDEs such as Xcode and Microsoft Visual Studio (tested on Visual C++ 2010).

4.1 Dependencies

Verdandi requires:

- the software construction tool SCons (version 1.0 or higher) for compilation;
- Python for both SCons and the optional Python interface (see page Python Interface) to the C++ code; the generation of the interface also requires SWIG.

Verdandi may require other dependencies:

- The Fortran libraries Blas and Lapack for linear algebra.
- The Newran library for random number generation.
- The NLopt library as optimization solver.

Note that in Verdandi tarball, you will also find:

- the linear algebra library Seldon.
- Lua, a scripting language.
- Ops, a library for reading Lua configuration files.

4.2 Linux

4.2.1 Installation

Download the source code (Verdandi homepage), usually available in a compressed file, e.g., verdandi-[version].tar.bz2. Uncompress the file, e.g., in command line: tar -xvjf verdandi-[version].tar.bz2. This will create the directory verdandi-[version]/ in which you will find Verdandi.
4.2.2 Installation of dependencies

Installations with APT (Advanced Package Tool from Debian and Ubuntu):

- Blas and Lapack (highly recommended) can be installed using the command-line:
  
  \$ sudo apt-get install libblas-dev liblapack-dev

- If you want to use the Python interface, install the following packages:
  
  \$ sudo apt-get install python-dev python-numpy swig

Manuals installations:

- If you want to use NLopt (for example in the clamped bar example), you will need to download and install it:
  
  \$ wget http://ab-initio.mit.edu/nlopt/nlopt-2.4.1.tar.gz
  \$ tar xvzf nlopt-2.4.1.tar.gz
  \$ cd nlopt-2.4.1
  \$ ./configure
  \$ make
  \$ sudo make install

- If you want to use Newran (for example in the shallow water model), you will need to download and install it:
  
  \$ wget http://www.robertnz.net/ftp/newran03.tar.gz
  \$ tar zxvf newran03.tar.gz

If everything is fine, you should have a file called include/newran/newran.h. Next, you have to edit include/newran/include.h in order to uncomment the line:

  //#define use_namespace

Then you can compile the library using:

  \$ make -f nr_gnu.mak libnewran.a

4.2.3 Testing the installation

In order to start with Verdandi, it is sufficient to install SCons. Make sure that the executable scons is available.

To compile one of the examples provided with Verdandi, run scons in the directory example/quadratic_model/:

  \$ scons

Then to run the quadratic model example, execute the following commands:

  \$ forward configuration/truth.lua [1.]
  \$ optimal_interpolation configuration/assimilation.lua [2.]

This should generate results in the form of .bin files found in the directory example/quadratic_model/result/.

1. This program generates the observations by running the model with the true initial conditions described in truth.lua, without any error. It performs steps forward with the quadratic model without data assimilation.

2. This program applies the optimal interpolation, starting from erroneous initial conditions described in assimilation.lua. The observations are those generated above by forward.
4.3 MacOS

4.3.4 Compile Verdandi as a library

Verdandi can be compiled as a library. In order to do so, you need to compile Seldon and Ops as libraries. To compile Seldon, you need to go in the Seldon directory, and then use the commands:

```
cd lib
scons lib=yes libseldon.a
```

Ops can be compiled using this command in the root directory of Ops:

```
scons lib=yes libops.a
```

If you include “share/SConstruct” from Verdandi, you can compile Verdandi with:

```
scons lib=compile libverdandi.a
```

Now, you can compile using Verdandi as a library using:

```
scons lib=yes
```

4.3 MacOS

4.3.1 Installation

Installation instructions for MacOS and Linux are nearly identical, except for a slight difference about the location of the directory where you have to put the files.

4.3.2 Create a Verdandi Xcode project

- Download Verdandi and expand it to the directory of your choice (MY_VERDANDI_PATH).
- Create a Verdandi project within Xcode (“File -> New Project”). In “Xcode/New Project”, select “Other/External Build System”. Choose a name (verdandi) and select the path to the source directory (MY_VERDANDI_PATH). In “Groups & Files”, right click project name (verdandi), then choose “Add -> Existing Files...” and add recursively the Verdandi project directories.
- Compiling under Xcode:
  1. In “Groups and Files -> Targets”, right click “Targets” then select “Add -> New Target...”, choose “other/External Target” and choose a name (for instance, forward).
  2. In “Groups and Files -> Targets”, double click the target that was created (forward).
  3. In the “Build Tool” field, put the full path to scons (for example, “/usr/local/bin/scons”).
  4. Set the “Directory” field to the directory that contains the SConstruct file (for example, MY_VERDANDI_PATH/example/clamped_bar/).
  5. In “Build Settings”, specify the architectures to which the binary is targeted (for instance, “ARCHS = x86_64”).
  6. You should now be able to build using the “Build” command from Xcode.
  7. Right click “Executables”, choose “Add new custom executable...”, make it point to the executable you are building, define the arguments (“configuration/truth.lua”) and then you can debug using Xcode.

4.4 Windows

4.4.1 Installation of the dependencies

Download precompiled libraries

Not all the libraries are needed for all the methods, select and install the ones you will want to use.
– Lua (always required). On this website, download the latest executable for version 5.1 which will install Lua on your computer. Alternatively, you may want to compile Lua yourself: in this case, you will need to install make and to compile the library which can be found in "verdandi/include/lua" inside Verdandi tarball.

– Blas and Lapack (highly recommended):
  – On this webpage, download the prebuilt libraries and extract the archive to some directory.
  – Add the path to this directory to the environment variable "path": "Computer" -> "Properties" -> "Advanced System Properties" -> "Environment variables".

– Atlas and Cblas (highly recommended): On this webpage download the file atlas330_WinNT_P4-SSE2.zip. Extract the archive to a directory.

4.4.2 Creation of a Visual Studio project (tested with Visual Studio 9 2008)

– Download and expand the Verdandi tarball.
– Create a new Visual Studio project from existing code: "File" -> "New" -> "Project From Existing Code".
– Select "Visual C++" in the drop-down menu then click "next".
– Set the project file location (for example "example/quadratic_model"), and give a name to the project, then click "next".
– In the drop-down menu select "Console application project" then click "finish".

4.4.3 Setting the dependencies in the Visual Studio project

– In "Project" -> "Properties" -> "Configuration Properties" -> "Additional Include Directories" add ".\..\", ".\..\include" and the path to the headers of your Lua installation.
– Still in "Configuration Properties" go to "linker" -> "General" in the menu on the left. Add the paths to lua, cblas and Blas/Lapack into "Additional Libraries Directories".
– Under "linker" in the "Input File" menu, add the librairies to the "Additional Dependencies" menu. (At this time the names are "cbia.lib.blas.dyn.rel.x86.12.lib", "cbia.lib.lapack.dyn.rel.x86.12.lib", "cblas.a", "lua51.lib")

4.4.4 Building the project and running a test - Quadratic model example.

– The first step is to select the "forward.cpp" file in the project file list. To do so, one can right click on all other .cpp file and "exclude from project".
– To run the project using Visual Studio, you need to define an output directory and a command line argument. The argument for "forward.cpp" is the path to the "configuration/truth.lua" in the quadratic example directory.
– At this point you can build the project using the build command of Visual Studio then run it.
– Forward will generate a set of observations which can be used in an assimilation method.
– Include the nudging "file.cpp" in the project, and exclude the "forward.cpp".
– The command line argument must now be set to "configuration\assimilation.lua".
– Rebuild and run this new build.
Chapter 5

Compilation

SCons is a software construction tool, used for building every example of Verdandi. It uses Python scripts as configuration files.

5.1 SCons Main Strengths

For compilation, Verdandi uses SCons (http://www.scons.org/). It relies on SConstruct files (similar to the makefiles) that are written in Python.

5.2 Basic Usage

In order to start with Verdandi, it is strongly recommended to install SCons. So make sure that the executable scons is available.

For example, to compile programs of the quadratic model, run scons in the directory example/quadratic_-model/:

$$ scons $$

If you want to compile only the program forward.cpp run:

$$ scons forward $$
Chapter 6

Example Programs

All examples are located in the example directory.

6.1 Shallow Water

The Shallow Water model describes the flow below a pressure surface in a fluid. The different boundary and initial conditions are described in the configuration files in example/shallow_water/configuration. First compile the examples with

$ scons

in the directory example/shallow_water. This will compile an example with ForwardDriver and an example with OptimalInterpolation (see the documentation of optimal interpolation).

6.1.1 Observations

Since no observations are given yet, we have to generate some. Execute the following command:

$ forward configuration/truth.lua

This will run the model with the initial conditions described in truth.lua, without data assimilation. This should generate a few result files in the directory example/shallow_water/result/. The water height is stored in the file truth-forecast_state.bin, and the horizontal velocity along x and y in respectively model-u.bin and model-v.bin. These files store time trajectories.

The generated state (water height) will serve as observations for the assimilation. To plot it, you can use IPython with the module pylab. Refer to the Python Interface section to make sure you can use the Python interface of Verdandi.

The following code will plot the water height at time t=0 and t=10.

$ ipython -pylab

In [1]: from verdandi import *
In [2]: truth = load_vector_list("result/truth-forecast_state.bin")
In [3]: plot(truth[0], label = "time t=0")
In [4]: plot(truth[10], label = "time t=10")
In [5]: legend()
In [6]: title("t = 10")

This should be the output:

The initial condition is a rectangular function in the center and the boundary condition is a flow from the left side. These conditions are described in example/shallow_water/configuration/shallow_water.lua.
6.1.2 Data assimilation with optimal interpolation

The observations are here managed by the linear observation manager. The observation operator is defined by:

```
for i = 1, 100 do
    for j = 1, 3 do
        observation.operator.value[(i-1)*3 + j] = 0.
    end
end
observation.operator.value[80] = 1.
observation.operator.value[100 + 81] = 1.
observation.operator.value[200 + 82] = 1.
```

This restricts the observations to three components of the water height vector with indexes 79, 80 and 81. See the Linear Observation Manager linear documentation about LinearObservationManager for further details. Note that the indexes in Lua structures start at 1 while the indexes in the rest of Verdandi (Python, C++, documentation, integers given in the configuration) start at 0.

To use the Optimal Interpolation method, execute the following command.

```
$ optimal_interpolation configuration/assimilation.lua
```

This runs the model with the initial conditions described in example/shallow_water/configuration/assimilation.lua. This should generate several files in the directory result/. The analysis states $x_a$ and the forecasted state $x_f$ in oi-forecast_state.bin and the horizontal velocity along x and y in respectively result/u.bin and result/v.bin.

We can now observe the results of data assimilation.

```
$ ipython -pylab
```

```
In [1]: from verdandi import *
In [2]: truth = load_vector_list("result/truth-forecast_state.bin")
In [3]: oi_forecast_state = load_vector_list("result/oi-forecast_state.bin")
In [4]: oi_analysis_state = load_vector_list("result/oi-analysis_state.bin")
In [5]: x = 10; plot(truth[x]); plot(oi_forecast_state[x]); plot(oi_analysis_state[x])
In [6]: legend(["truth", "forecast", "analysis"]); title("t = 10")
```

This should plot:

Since we have restricted the observations to the points $x = 79, 80$ and $81$, and since the observations have the same values as the forecast at these locations (that is, 1), the increment $y_h - H_h x_h$ is zero and the assimilation has no impact.

Let us take a look at a later time step. In the same ipython console, type:

```
In [7]: clf()
In [8]: x = 20; plot(truth[x]); plot(oi_forecast_state[x]); plot(oi_analysis_state[x])
In [9]: legend(["truth", "forecast", "analysis"]); title("t = 20")
```

for the following result:

Here, the observed wave originating from the rectangular initial condition has reached the observed locations. This produces an analysis which corrects the forecasted state $x_f^h$ (in green) according to the observations. This analysis $x_a^h$ is then reinjected in the model for the next time step $h + 1$.

More pronounced at $t=30$:

```
In [10]: clf()
In [11]: x = 30; plot(truth[x]); plot(oi_forecast_state[x]); plot(oi_analysis_state[x])
In [12]: legend(["truth", "forecast", "analysis"]); title("t = 30")
```

Let us change the parameters in the configuration files and see the effects. For example, in example/shallow_water/configuration/shallow_water.lua, change the value of shallow_water.state_error.variance from 100 to 25.

Execute optimal_interpolation once again and visualize the results:
Since we lowered the variance of the state error (diagonal of \( B_h \)), the analysis stays closer to the forecast.

### 6.2 Clamped Bar

The Clamped Bar model describes the vibration of a bar clamped at one end. The boundary and initial conditions are described in the configuration files found in `example/clamped_bar/configuration`.

First compile the examples with

```bash
$ scons
```

in the directory `example/clamped_bar`. This will compile an example with each of the data assimilation methods `ForwardDriver`, `OptimalInterpolation` (optimal interpolation), `ExtendedKalmanFilter` (extended Kalman filter), `UnscentedKalmanFilter` (unscented Kalman filter), `ReducedOrderExtendedKalmanFilter` (reduced order extended Kalman filter), `ReducedOrderUnscentedKalmanFilter` (reduced order unscented Kalman filter), and `FourDimensionalVariational` (four dimensional variational).

#### 6.2.1 Observations

Since no observations are given yet, we have to generate some. Execute the following command:

```bash
$ forward configuration/truth.lua
```

to run the model with the initial conditions described in `truth.lua`, without data assimilation. This should generate a result file (`result/truth-forecast_state.bin`) in the directory `example/clamped_bar/result/`. This file store the state (displacement, velocity, \( \theta_f \)) trajectory.

The generated state (displacement, velocity, \( \theta_f \)) will serve as observations for the assimilation. To plot it, you can use IPython with the module pylab. Refer to the Python Interface section to make sure you can use the Python interface of Verdandi.

The following code will plot the displacement at time \( t=0 \) and \( t=10 \).

```bash
$ ipython -pylab
```

The output should look something like this:

At the initial time, the bar is fixed and horizontal.

During the simulation, a force \( F(\theta_f) = \sin(\frac{\pi}{\theta_f}) M_{\theta_f}(1...1)^T \) is applied to the bar at each point. For the parameter \( \theta_f \), we supposed that the bar is divided into two regions of the same length. In the first region ([0, 4]), we have \( \theta_f = 1.5 \) while, in the second region ([5, 9]), we have \( \theta_f = 1.7 \). These conditions are described in `example/clamped_bar/configuration/clamped_bar.lua`.
6.2.2 Data assimilation with ROEKF and ROUKF.

The observations are managed by the linear observation manager. The parameters for the observations are stored in example/clamped_bar/configuration/observation.lua. By default, the observation operator is a scaled identity matrix with diagonal values of 1, which means that the whole state is observed. For a more illustrative example, we can choose to restrain the observations to specific components of the state.

In the file example/clamped_bar/configuration/observation.lua, change the parameter observation.operator.scaled_identity from true to false and put the following code at the end of the file:

```lua
Nstate = 22
Nobservation = 10
for i = 1, Nobservation * Nstate do
    observation.operator.value[i] = 0.
end
for i = 1, Nobservation do
    for j = 1, Nobservation do
        if i == j then
            observation.operator.value[Nstate * (i - 1) + j] = 1.0
        end
    end
end
```

This will restrict the observations to the displacements. See the Linear Observation Manager documentation for further details. Note that the indexes in Lua structures start at 1 while the indexes in the rest of Verdandi (Python, C++, documentation, integers given in the configuration) start at 0.

To use the Reduced Order Extended Kalman Filter and the Reduced Order Unscented Kalman Filter methods, execute the following commands.

```
$ reduced_order_extended_kalman_filter configuration/assimilation.lua
$ reduced_order_unscented_kalman_filter configuration/assimilation.lua
```

This runs the model with the initial conditions described in example/clamped_bar/configuration/assimilation.lua. The simulation begins with erroneous values for the parameter $\theta_f$. This should generate several files in the directory result/. The analysis states $x^a_H$ are stored in the files result/roekf-analysis_state.bin and result/roukf-analysis_state.bin, the forecasted state $x^f_H$ in roekf-forecast_state.bin and roukf-forecast_state.bin.

We can now observe the results of data assimilation.

```
$ ipython -pylab
```

```
In [1]: from verdandi import *
In [2]: N = 22
In [3]: roekf = loadtxt("result/roekf-analysis_state.dat", usecols = range(0, N))
In [4]: roukf = loadtxt("result/roukf-analysis_state.dat", usecols = range(0, N))
In [5]: x = 20; plot(roekf[:, x]); plot(roukf[:, x]);legend(("roekf", "roukf"), "lower right");
In [6]: clf()
In [7]: x = 21; plot(roekf[:, x]); plot(roukf[:, x]);legend(("roekf", "roukf"), "lower right");
```

This should plot:

Since the Clamped Bar model is linear with respect to the parameter $\theta_f$, the Reduced Order Extended Kalman Filter and the Reduced Order Unscented Kalman Filter methods give the same results. In this simulation, we start from erroneous values of the parameter $\theta_f = (1.0, 1.0)$ and the methods converge to the correct values $(1.5, 1.7)$. 

---

Generated on Mon Sep 28 2015 16:49:58 for Verdandi by Doxygen
Chapter 7

Notation

Unless otherwise specified, the notation used throughout the whole documentation (including the reference documentation) is that described below.

For details about the C++ notation of the variables involved in data assimilation algorithms, see section reference notation.

7.1 Model and State Vectors

The time index $h$ is associated with the time $t_h$. The model time steps are denoted $\Delta t_h$, so that $t_h = t_0 + \sum_{h'=0}^{h-1} \Delta t_{h'}$.

The state equation is

$$x_{h+1} = M_h(x_h, p_h) + e_m,$$

where $x_h \in \mathbb{R}^n$ is the true model state vector, $M_h$ is the model operator, $p_h \in \mathbb{R}^l$ is the vector of model parameters, and $e_m$ is the model error.

The model error can be decomposed into a systematic error $\bar{e}_m$ and a fluctuation $\tilde{e}_m$:

$$e_m = \bar{e}_m + \tilde{e}_m.$$

If the model error is a random vector, then $\bar{e}_m$ is its expectation: $\bar{e}_m = E(e_m)$.

A tangent linear model is denoted $M_h$.

In the course of a simulation with data assimilation, one distinguishes the forecast state vector $x_f$ and the analysis state vector $x_a$. The latter is the result of the assimilation, while the former is often defined as

$$x_{f+1} = M_h(x_f, p_h).$$

The forecast and analysis errors are defined as

$$e_f = x_f - x_h,$$

$$e_a = x_a - x_h.$$

The variances of the errors are denoted $P_f$ for the forecast error, $P_a$ for the analysis error and $Q$ for the model error:

$$P_f = E\left((e_f - E(e_f)) (e_f - E(e_f))^T\right),$$

$$P_a = E\left((e_a - E(e_a)) (e_a - E(e_a))^T\right),$$

$$Q = E\left(\bar{e}_m (\bar{e}_m)^T\right).$$
7.2 Observations

The observation vector at time $t_h$ is denoted $y_h \in \mathbb{R}^m$. It satisfies

$$y_h = \mathcal{H}_h(x_f^h) + e_h^o$$

The observation operator $\mathcal{H}_h$ maps the model state space into the observation space. A tangent linear operator is denoted $H_h$.

The observation error can be decomposed into a systematic error $\bar{e}_h^o$ and a fluctuation $\tilde{e}_h^o$:

$$e_h^o = \bar{e}_h^o + \tilde{e}_h^o,$$

with $\bar{e}_h^o = \mathbb{E}(e_h^o)$ in a stochastic framework.

The observational error covariance matrix is

$$R_h = \mathbb{E}\left((\tilde{e}_h^o)^T (\tilde{e}_h^o)\right).$$

The innovation, defined as the discrepancy between the observations and the corresponding model forecast, is denoted

$$d_h = y_h - \mathcal{H}_h(x_f^h).$$
Chapter 8

Overview

Below is a summary of Verdandi contents.

8.1 Methods

In the directory `method/`, several data assimilation methods:

- optimal interpolation (OI);
- extended Kalman filter (EKF);
- reduced order extended Kalman filter (ROEKF);
- unscented Kalman filter (UKF);
- reduced order unscented Kalman filter (ROUKF);
- reduced minimax filter (RMF);
- four dimensional variational (4DVAR);
- ensemble Kalman filter (EnKF).

Source code

```cpp
class OptimalInterpolation<T, ClassModel, ClassObservationManager>
OptimalInterpolation.hxx
OptimalInterpolation.cxx
```

```cpp
class ExtendedKalmanFilter<T, ClassModel, ClassObservationManager>
ExtendedKalmanFilter.hxx
ExtendedKalmanFilter.cxx
```

```cpp
class UnscentedKalmanFilter<T, ClassModel, ClassObservationManager>
UnscentedKalmanFilter.hxx
UnscentedKalmanFilter.cxx
```

```cpp
class ReducedOrderExtendedKalmanFilter<T, ClassModel, ClassObservationManager>
ReducedOrderExtendedKalmanFilter.hxx
ReducedOrderExtendedKalmanFilter.cxx
```
class ReducedOrderUnscentedKalmanFilter<T, ClassModel, ClassObservationManager>
ReducedOrderUnscentedKalmanFilter.hxx
ReducedOrderUnscentedKalmanFilter.cxx
class BasePerturbationManager
BasePerturbationManager.hxx
BasePerturbationManager.cxx

8.2 Models

In the directory model/, several sample models:

- Quadratic Model;
- Shallow-water;
- Clamped Bar;
- Lorenz Model.

Source code

class ClampedBar<T>
ClampedBar.hxx
ClampedBar.cxx
class ShallowWater<T>
ShallowWater.hxx
ShallowWater.cxx
class Lorenz<T>
Lorenz.hxx
Lorenz.cxx
class QuadraticModel<T>
QuadraticModel.hxx
QuadraticModel.cxx

8.3 Observation managers

In the directory observation_manager/, two (linear) observation managers:

- a linear observation manager whose observation operator is simply defined with a matrix;
- a grid to network observation manager with an observation operator that maps from a model grid to an observation network;
- an observation aggregator.
8.4 Miscellaneous

In the directory *output_saver/*, one may find a class to save several kinds of output results.

Source code

```cpp
class LinearObservationManager<T> 
LinearObservationManager.hxx
LinearObservationManager.cxx

class GridToNetworkObservationManager<T> 
GridToNetworkObservationManager.hxx
GridToNetworkObservationManager.cxx

class ObservationAggregator<T> 
ObservationAggregator.hxx
ObservationAggregator.cxx
```

8.4 Miscellaneous

In the directory *output_saver/*, one may find a class to save several kinds of output results.

Source code

```cpp
class OutputSaver 
OutputSaver.hxx
OutputSaver.cxx
```

In the directory *share/*, one may find a class for objects to communicate.

8.5 Scripts

Useful scripts may be found in the directory *bin/*.

The script *format* partially formats source files according to Verdandi’s rules: it removes trailing spaces and indent the code; it finally checks that no line goes beyond the 78th column.

8.6 Verdandi directories

Verdandi directories are the following:

- *bin/*: useful scripts;
- *error/*: error variances;
- *example/*: example programs for several models and configurations;
- method/: the data assimilation algorithms;
- model/: several example models;
- observation_manager/: example observation managers;
- output_saver/: a convenient class to save (on disk) different kinds of variables;
- share/: functionalities shared between the objects, such as the handling of messages, the error management and some useful functions.
Chapter 9

Configuration Files

9.1 Ops in short

Ops is a C++ library and a Python module for reading configuration files. The files are written in Lua, which makes the configuration files easy to read and write for the beginner, and yet very powerful for advanced users.

Ops supports reading Boolean, integers, floats, doubles and strings, and vectors (of any of the previous types). It is also possible to call functions defined in the configuration files.

The user can define constraints to be satisfied by the entries (e.g., positive integer, list of acceptable values, ...). In case of errors, exceptions are raised.

Ops is capable of saving the Lua definitions of all variables (except functions) that have been read in a configuration file. This may be useful to keep track of configurations.

Ops only depends on Lua 5.1. Both Ops and Lua 5.1 are provided in the Verdandi tarball.

9.2 Example

The main features of the library are illustrated by the following example. No exception is raised in this example, but Ops raises an exception any time an error is detected (missing file, missing entry, constraint not satisfied, Lua interpretation error, ...) and shows an information message to help the user.

Configuration files must be written in Lua 5.1 and are read by the library Ops. Here is an example file:

eexample.lua :  

```
-- Lines starting with "--" are comment lines.

-- This defines an integer.
Nx = 100
-- Calculations are possible.
Ny = 2 * Nx - 10

-- This defines a floating number in double precision.
Delta_t_model = 0.03
-- Another example calculation:
Delta_t_sqrt = math.sqrt(Delta_t_model) * 1.5e-7

-- Here is a string.
output_directory = "result/
-- Concatenation of strings is carried out by ".
output_file = output_directory .. "output.bin"
```

----------------------------------- MODEL -----------------------------------

-- Configuration entries are provided in sections and subsections.
model = {
    definition = {
        -- You can provide a vector.
        initial_state = {0.3, 0.3},

        -- Binary options (Boolean).
        with_linear_term = true,
        with_constant_term = false,

        -- Reference to a previously defined entry.
        Delta_t = Delta_t_model
    }
}

-- Entries can be overwritten:
model.definition.initial_state = {1., 1.6}

advanced_example.lua :

-- A list of vectors is provided as a single vector (which will be split by
-- Ops).
vector_list = {u_1, u_2, u_3, v_1, v_2, v_3, w_1, w_2, w_3}

-- Note that the following syntax is absolutely equivalent (Lua does not pay
-- attention to line breaks).
vector_list = {u_1, u_2, u_3, v_1, v_2, v_3, w_1, w_2, w_3}

-- A list of matrices is also provided as a single vector (which will be split
-- by Ops).
matrix_list = {u_11, u_12, u_21, u_22, v_11, v_12, v_21, v_22}

vector_list represents three vectors defined as:

\[
\begin{pmatrix}
  u_1 \\
  u_2 \\
  u_3
\end{pmatrix}^T,
\begin{pmatrix}
  v_1 \\
  v_2 \\
  v_3
\end{pmatrix}^T,
\begin{pmatrix}
  w_1 \\
  w_2 \\
  w_3
\end{pmatrix}^T
\]

matrix_list represents two matrices:

\[
\begin{pmatrix}
  u_{11} & u_{12} \\
  u_{21} & u_{22}
\end{pmatrix},
\begin{pmatrix}
  v_{11} & v_{12} \\
  v_{21} & v_{22}
\end{pmatrix}
\]
Chapter 10

Debugging

10.1 Compilation options

To enable the -g compilation option, you can use the debug option with SCons. The possible values for this variable are

- `-1`: no optimization or debugging compilation option; this is for fast compilation;
- `0` (the default value): -O2
- `1`: -g
- `2`: -O2 -g

For example, a call to

```bash
g scons debug=2
```

will compile the source files with the options -O2 and -g.

The SCons option line enables to print the full command line used to compile your source files. To use it, launch

```bash
g scons line=yes
```

10.2 Dumping SCons environment

If you need to access the SCons environment variables, you may launch

```bash
g scons dump_env=yes
```

This will create a file `scons_env.log` that describes the entire SCons environment.

10.3 Exceptions

10.3.1 Options

Verdandi is able to check for several mistakes:

- `VERDANDI_CHECK_CONFIGURATION`: checks that there is no error in configurations.
- `VERDANDI_CHECK_IO`: checks input/output operations on disk.
– VERDANDI_CHECK_PROCESSING: checks some data processing.
– VERDANDI_CHECK_ARGUMENT: checks functions or methods arguments.
– VERDANDI_CHECK_DIMENSIONS: checks that the dimensions of involved structures are compatible. Notice that there are methods in which the compatibility is not checked however.

Seldon is also able to check for several mistakes:
– SELDON_CHECK_IO: checks input/output operations on disk.
– SELDON_CHECK_MEMORY: checks memory allocations and deallocations.
– SELDON_CHECK_DIMENSIONS: checks that the dimensions of involved structures are compatible. Notice that there are methods in which the compatibility is not checked however.
– SELDON_CHECK_BOUNDS: checks that indices are not out of range.

To enable VERDANDI_CHECK_IO, for example, put (before #include <Verdandi.hxx>):

```c
#define VERDANDI_CHECK_IO
```

Alternatively, there are debug levels:

– SELDON_DEBUG_LEVEL_0: nothing is checked.
– SELDON_DEBUG_LEVEL_1: equivalent to SELDON_CHECK_IO plus SELDON_CHECK_MEMORY.
– SELDON_DEBUG_LEVEL_2: equivalent to SELDON_DEBUG_LEVEL_1 plus SELDON_CHECK_DIMENSIONS.
– SELDON_DEBUG_LEVEL_3: equivalent to SELDON_DEBUG_LEVEL_2 plus SELDON_CHECK_BOUNDS.
– SELDON_DEBUG_LEVEL_4: equivalent to SELDON_DEBUG_LEVEL_3.
– VERDANDI_DEBUG_LEVEL_0: equivalent to VERDANDI_CHECK_CONFIGURATION plus VERDANDI_CHECK_ARGUMENT plus VERDANDI_CHECK_PROCESSING plus SELDON_DEBUG_LEVEL_0.
– VERDANDI_DEBUG_LEVEL_1: equivalent to VERDANDI_DEBUG_LEVEL_0 plus SELDON_DEBUG_LEVEL_1 plus VERDANDI_CHECK_IO.
– VERDANDI_DEBUG_LEVEL_2: equivalent to VERDANDI_DEBUG_LEVEL_1 plus SELDON_DEBUG_LEVEL_2 plus VERDANDI_CHECK_DIMENSIONS.
– VERDANDI_DEBUG_LEVEL_3: equivalent to VERDANDI_DEBUG_LEVEL_2 plus SELDON_DEBUG_LEVEL_3.
– VERDANDI_DEBUG_LEVEL_4: equivalent to VERDANDI_DEBUG_LEVEL_3 plus SELDON_DEBUG_LEVEL_4.

In practice, it is advocated to choose VERDANDI_DEBUG_LEVEL_4 in the development stage and VERDANDI_DEBUG_LEVEL_2 for the stable version. Indeed VERDANDI_DEBUG_LEVEL_4 slows down the program but checks many things and VERDANDI_DEBUG_LEVEL_2 should not slow down the program and ensures that it is reasonably safe.

Development stage:

```c
#define VERDANDI_DEBUG_LEVEL_4
```

Stable version:

```c
#define VERDANDI_DEBUG_LEVEL_2
```
10.3.2 Exceptions raised

The objects that may be launched by Verdandi are of type: ErrorConfiguration, ErrorIO, ErrorProcessing, ErrorUndefined, ErrorArgument. They all derive from Error. They provide the method What() that returns a string explaining the error, and the method CoutWhat() that displays on screen this explanation.

10.3.3 Exceptions and debugging

Suppose your code contains an error and raises an exception. You probably want to identify the function that raised the exception. The error message should contain the name of the function. But you probably want to know the exact line where the error occurred and the sequence of calls. Then, you have two options, using a debugger.

One option is to place a breakpoint in Error::Error(string function = "", string comment = "") (see file share/Error.cxx) because this constructor should be called before the exception is actually raised.

Another option, more convenient because no breakpoint is to be placed, is to define VERDANDI_WITH_ABORT. With that flag activated, if a Verdandi exception is raised, the program will simply abort. The call stack is then at hand.

10.4 Logger

Verdandi offers the ability to write messages in a log file or in the standard output thanks to its static class Verdandi::Logger.

10.4.1 Logger status

It is possible to change the logger status using the methods Verdandi::Logger::Activate() and - Verdandi::Logger::Deactivate(). These methods enable to activate or deactivate the logger.

VERDANDI_LOG_IS_ACTIVE specifies the logger status (active by default).

To deactivate the logger, put before #include <Verdandi.hxx>:

#define VERDANDI_LOG_IS_ACTIVE false

10.4.2 Logger file name

VERDANDI_LOG_FILENAME defines the name of the log file (verdandi.log by default).

To change the log file name, for example, put before #include <Verdandi.hxx>:

#define VERDANDI_LOG_FILENAME "verdandi-%{D}.log"

The special character %{D} will be replaced by the current date.

It is also possible to define the log file in a configuration file, if the method Verdandi::Logger::Initialize(string file_name, string section_name) is called. For instance:

Logger::Initialize("configuration.cfg", "logger/");

Then, in order to set the path of the log file, you have to define or to change the variable File in the section logger of the configuration file configuration.cfg.
10.4.3 Two methods

The logger mainly comes with the methods `Verdandi::Logger::Log()` and `Verdandi::Logger::StdOut()`.

`Verdandi::Logger::Log()` writes the messages in the log file, and also on the standard output when the messages are important enough. `Verdandi::Logger::StdOut()` writes the messages on the standard output, and in the log file too (unless configured otherwise).

10.4.4 Logger options

It is possible to tune where and how the messages are actually written:

- Usually, one wants the messages to appear in the log file. It is however possible to deactivate this with the method `Verdandi::Logger::SetFile(bool)`. After the call `Logger::SetFile(false)`, the logger will not write the messages to the log file.

- `Verdandi::Logger::StdOut()` will always print its messages on the standard output. In the default setting, `Verdandi::Logger::Log()` will write the important messages on the standard output (see below the subsection about priorities). If you want that `Verdandi::Logger::Log()` writes all messages on the standard output, use `Verdandi::Logger::SetStdout(bool)`.

- The method `Verdandi::Logger::SetUppercase(bool)` determines whether the messages will be written in uppercase or not. The default behavior is not to write the log messages in uppercase.

Instead of calling the three previous methods, it is possible to set up the corresponding options with `VERDANDI_LOG_OPTIONS`. These options are encoded using the constants called `Verdandi::Logger::file_`, `Verdandi::Logger::stdout_` and `Verdandi::Logger::upper_case_`. The default value of `VERDANDI_LOG_OPTIONS` is

```c
#define VERDANDI_LOG_OPTIONS Verdandi::Logger::file_
```

If you want to deactivate the log file, and activate the two other options (standard output and uppercase), then put the following definition before `#include <Verdandi.hxx>`:

```c
#define VERDANDI_LOG_OPTIONS Verdandi::Logger::stdout_ | Verdandi::Logger::upper_case_
```

10.4.5 Formatting message

`VERDANDI_LOG_WIDTH` defines the number of characters per line (78 by default).

10.4.6 Level of priority

A level of priority may be associated to each call of `Verdandi::Logger::Log()` using a template parameter. Only messages whose priority levels are greater than or equal to the level of verbosity are actually written.

`VERDANDI_LOGGING_LEVEL` defines the level of verbosity (0 by default).

`VERDANDI_EXCEPTION_LOGGING_LEVEL` defines the priority level for exception messages (15 by default).

When a message is written on the standard output, it is also written in the log file if `VERDANDI_STDOUT_LOGGING_LEVEL` is greater than or equal to the logging level. In the default setting, this is the case since `VERDANDI_STDOUT_LOGGING_LEVEL` is set to 7.

10.4.7 Example

```c
#define VERDANDI_LOG_IS_ACTIVE false
#define VERDANDI_LOG_FILENAME "verdandi-%{D}.log"
```
#include "Verdandi.hxx"

using namespace Verdandi;

#include "Logger.cxx"

class ClassTest
{
  public:

    string GetName() const
    {
      return "ClassTest";
    }

    void MemberFunction()
    {
      Logger::Log(*this, "ok");
    }
};

int main(int argc, char** argv)
{
  TRY;

  Logger::Log<5>("TEST 1", "ok");
  Logger::Activate();
  Logger::Log<5>("TEST 2", "ok");
  Logger::SetOption(Logger::stdout_ | Logger::file_, true);
  Logger::Log<5>("TEST 3", "ok");
  Logger::Log<-5>("TEST 4", "ok");
  Logger::Command("hline", "-", Logger::file_);
  Logger::InitializeOptions();
  ClassTest test;
  test.MemberFunction();

  END;

  return 0;
}
Chapter 11

Python Interface

Verdandi comes with a Python interface generated by Swig. This interface exposes C++ classes in Python. This is useful for interactive use, for postprocessing (e.g., for easy access to the observations provided by the observation manager), maybe to debug, and for people that would like to implement data assimilation methods in Python.

This page does not deal with the use of a model or an observation manager written in Python. If the model or the observation manager are themselves in Python, you should refer to the page Plugging a Python model, which describes Verdandi facilities to interface Python to C++.

This page only addresses the compilation and the use of the interface under Linux, MacOS and Windows. The generation of the interface was not tested on another platform yet. No known issue should prevent the interface from running successfully on another platform.

It is recommended to use IPython (enhanced Python shell) instead of the regular Python shell, with Matplotlib, a 2D plotting library which produces high quality figures. Once installed, launch IPython's pylab mode with ipython -pylab. You should get the prompt

In [1]:

If no error occurs, you have successfully installed Matplotlib. You can try to plot some figures with the command plot.

In [1]: plot([0,1])

This should open a new window with the figure of a line from (0,0) to (1,1). The prompt should still be available, so you can plot other figures.

In [2]: plot([4,-1])

Another figure has been added, a line from (0,4) to (1,-1).

In addition to a C++ compiler, one needs Swig 1.3.x. Swig 1.1 cannot generate the interface. You will also need Python (say, 2.5 or 2.6, but previous versions should work too) and its headers.

In python/ directory, you may simply launch scons swig if you have SCons installed (version $>$=1.0).

This should generate the Python module verdandi.py and the shared library _verdandi.so. You may want to place these two files in a directory of your PYTHONPATH, where Python searches for modules.

Swig is available for Windows. Download the prebuild executable, and add the path to SWIG to your environment variable. Create a "New project from existing code file" in Visual. Select Visual C++ for the entry "Type of project". Specify then the location of the "python" directory in your version of Verdandi: Verdandi-[version]\python, and give a name to the project. On the "Project Settings" page, select "Use external build system". Then put scons on the generic build command line and scons -c on the clean command line. Click on OK.

Then add the different library paths needed, in the entry "Library Directories" (in "Project -> Properties -> Configuration Properties -> VC++ Directories"). The default SWIG interface of Verdandi has dependencies to Python, Lua, Blas, Lapack and Cblas.
Build the solution, which will generate the Python module `verdandi.py` and the shared library `_verdandi.pyd`. You may want to place these two files in the directory of your DLLs in Python (by default: `C:\Python\DLLs`), where Python searches for modules.

To manipulate vectors from the Python interface, you need Seldon module. To build this module, run `scons` from `seldon/src` directory. This will build the Swig interface to Python:

```
$ scons
```

`seldon.py` and `_seldon.so` have been generated. Make sure these files are in a directory of `$PYTHONPATH` to be able to launch Seldon Python module.

Create a "New project from existing code file" in Visual. Select Visual C++ for the entry "Type of project". Specify then the location of the Seldon directory in your version of Verdandi: `Verdandi-[version]\include\seldon`, and give a name to the project. On the "Project Settings" page, select "Use external build system". Then put `scons` on the generic build command line and `scons -c` on the clean command line. Click on OK.

Then add the different library paths needed, in the entry "Library Directories" (in "Project -> Properties -> Configuration Properties -> VC++ Directories"). Seldon only needs a link to the Python library, so put the path to the `libs` directory of your Python installation.

Build the solution, which will generate in the Seldon directory the Python module `seldon.py` and the shared library `_seldon.pyd`. You may want to place these two files in the directory of your DLLs in Python (by default: `C:\Python\DLLs`), where Python searches for modules.

From `python/` directory, or from any place if `verdandi.py` and `_verdandi.so` (or `_verdandi.pyd`) are in a directory of `$PYTHONPATH` under Linux and MacOS or in Python DLLs directory under Windows, you may launch Python and load the module.

```
$ ipython -pylab
```

```
In [1]: import seldon
In [2]: import verdandi

In [3]: method = verdandi.Method()
In [4]: method.Initialize("configuration_file.lua")

In [5]: model = method.GetModel()
In [6]: model.GetTime()
Out [6]: 0
```

Forward() can be processed either by calling the driver method or directly by calling the model method:

```
In [7]: method.Forward()
In [8]: model.GetTime()
Out [9]: 0.0015
In [10]: model.Forward()
In [11]: model.GetTime()
Out[12]: 0.0030
```

```
In [13]: for i in range(1008):
....:   model.Forward()
```

```
In [14]: model.GetTime()
Out[14]: 1.515
```

Here is an example of the computation of an analysis. You may want to check its effect by printing the state vector before and after the analysis:

```
In [15]: state_vector = seldon.VectorDouble()
In [16]: model.GetState(state_vector)
In [17]: state_vector.Print()
0 0
In [18]: method.Analyze()
In [19]: model.GetState(state_vector)
In [20]: state_vector.Print()
0.0272727 0.0272727
```

Here is an example of the computation of an analysis. You may want to check its effect by printing the state vector before and after the analysis:
Here is an example of the interactivity of the high-level interface: the first element of the state vector is set to an arbitrary value:

```python
In [21]: state_vector[0] = 0
In [22]: model.SetState(state_vector)
In [23]: model.GetState(state_vector)
In [24]: state_vector.Print()
0 0.0272727
```
Part III

Data Assimilation Methods
Chapter 12

Assimilation Methods

Below are the data assimilation methods available in Verdandi:

- optimal interpolation (OI);
- extended Kalman filter (EKF);
- reduced order extended Kalman filter (ROEKF);
- unscented Kalman filter (UKF);
- reduced order unscented Kalman filter (ROUKF);
- reduced minimax filter (RMF);
- four dimensional variational (4DVAR);
- ensemble Kalman filter (EnKF).

Not all assimilation methods are compatible with all example models, because the latter may not have an interface complete enough. The following table indicates which methods can be applied to which example models.

<table>
<thead>
<tr>
<th></th>
<th>Clamped Bar</th>
<th>Shallow Water</th>
<th>Lorenz</th>
<th>Quadratic Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal interpolation</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Extended Kalman filter</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Reduced order extended Kalman filter</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Unscented Kalman filter</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Reduced order unscented Kalman filter</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Reduced minimax filter</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Four dimensional variational</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Ensemble Kalman filter</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>
Chapter 13

Ensemble Kalman Filter

Verdandi provides a C++ implementation of the Ensemble Kalman Filter (EnKF).

The Ensemble Kalman Filter (EnKF) is implemented in EnsembleKalmanFilter.hxx and EnsembleKalmanFilter.cxx. The class EnsembleKalmanFilter is a template class: EnsembleKalmanFilter<T, ClassModel, ClassObservationManager, ClassPerturbationManager>. T is the type of the elements to be stored (e.g., double), ClassModel is the type of the model (e.g., ShallowWater<double>), ClassObservationManager is the type of the observation manager (e.g., LinearObservationManager<double>), and ClassPerturbationManager is the type of the perturbation manager (e.g., NewranPerturbationManager).

A simulation with Ensemble Kalman filter (EnKF) may be carried out with the following C++ lines:

```cpp
EnsembleKalmanFilter<real, ShallowWater<real>, LinearObservationManager<real>, NewranPerturbationManager> driver; [1]
driver.Initialize(argv[1]); [2]
while (!driver.HasFinished()) [6]
{
    driver.InitializeStep(); [3]
    driver.Forward(); [4]
    driver.Analyze(); [5]
}
```

1. First build the EnsembleKalmanFilter driver with the construction EnsembleKalmanFilter.
2. Then initialize the driver, the model, the observation manager, the perturbation manager and read option keys in the configuration file with Initialize(configuration_file). Model parameters or initial conditions are perturbed differently for each member of the ensemble at that stage.
3. Optionally initialize the model before a time step with InitializeStep(). Perturbations can be applied again.
4. Perform a step forward with Forward() for all members of the ensemble.
5. Compute the analysis with Analyze() whenever observations are available.
6. Iterate until the model has finished: HasFinished() returns true if the simulation is done, false otherwise.

13.1 Ensemble Kalman Filter algorithm

In the ensemble Kalman filter, the forecast error covariance matrix is estimated with an ensemble of simulations. Each member of the ensemble is defined with perturbations in the initial condition and in several uncertain input parameters. The filter estimator is given by the ensemble mean.
1. At time $t_0$, an ensemble $(x_{f,0}^{(i)})$ of initial conditions is generated. The uncertain parameters $p_0$ of the model are perturbed in each member, and denoted $\tilde{p}_0^{(i)}$.

2. For every time $t_h, h \geq 0$,
   - the forecast is the ensemble mean
     $$x_{f,h} = \frac{1}{N} \sum_{i=1}^{N} x_{f,h}^{(i)}$$
   - if observations $y_h$ are available, the forecast error covariance matrix is approximated by,
     $$P_{f,h} = \frac{1}{N-1} \sum_{i=1}^{N} (x_{f,h}^{(i)} - x_{h}) (x_{f,h}^{(i)} - x_{h})^T$$
     The analysis is computed for each member of the ensemble:
     $$x_{a,h}^{(i)} = x_{f,h}^{(i)} + K_h(y_h - H_h(x_{f,h}^{(i)})),$$
     with
     $$K_h = P_{f,h} H_h^T (H_h P_{f,h} H_h^T + R_h)^{-1}.$$  
     The subsequent forecast is
     $$x_{f,h+1}^{(i)} = \mathcal{M}_h(x_{a,h}^{(i)}, \tilde{p}_h^{(i)}).$$  
     The parameters $p_{h+1}$ are then perturbed and denoted $\tilde{p}_{h+1}$.
   - if there are no observations, the forecast is
     $$x_{f,h+1}^{(i)} = \mathcal{M}_h(x_{f,h}^{(i)}, \tilde{p}_h^{(i)}).$$  
     The parameters $p_{h+1}$ are then perturbed and denoted $\tilde{p}_{h+1}$.

With:
- $x_{f,h}$ forecast state vector (ensemble mean);
- $x_{a,h}$ analysis state vector (ensemble mean);
- $x_{f,h}^{(i)}$ forecast state vector of member $i$ of the ensemble;
- $x_{a,h}^{(i)}$ analysis state vector of member $i$ of the ensemble;
- $y_h$ observation vector;
- $\mathcal{H}_h$ observation operator that maps the state space to the observation space;
- $H_h^{(i)}$ observation operator linearized at $x_{f,h}^{(i)}$;
- $P_{f,h}$ error covariance matrix of $x_{f,h}^{(i)}$;
- $R_h$ observational error covariance matrix;
- $K_h$ analysis gain matrix;
- $p_h$ model parameters;
- $\tilde{p}_h$ perturbed model parameters;
- $\mathcal{M}_h$ model.
Chapter 14

Extended Kalman Filter

Verdandi provides a C++ implementation of the extended Kalman filter (EKF).

The extended Kalman filter (EKF) is implemented in ExtendedKalmanFilter.hxx and ExtendedKalmanFilter.cxx. The class ExtendedKalmanFilter is a template class: ExtendedKalmanFilter<T, ClassModel, ClassObservationManager>. T is the type of the elements to be stored (e.g. double), ClassModel is the type of the model (e.g. ClampedBar<double>), ClassObservationManager is the type of the observation manager (e.g. LinearObservationManager<double>).

A simulation with the Extended Kalman Filter (EKF) may be carried out with the following C++ lines:

```
ExtendedKalmanFilter<real, ClampedBar<real>, 
    LinearObservationManager<real> > driver;  // [1]
driver.Initialize(argv[1]);  // [2]
while (!driver.HasFinished())  // [6]
{
    driver.InitializeStep();  // [3]
    driver.Forward();  // [4]
    driver.Analyze();  // [5]
}
```

1. First build the ExtendedKalmanFilter driver with the construction ExtendedKalmanFilter.
2. Then initialize the driver, the model and the observation manager, and read option keys in the configuration file with Initialize(configuration_file). This optionally computes an analysis (BLUE) with the model initial condition.
3. Optionally intialize a step with InitializeStep(). This initializes a step for the model.
4. Perform a step forward and propagate the state error variance with Forward().
5. Compute the analysis with Analyze(), whenever observations are available.
6. Compute the data assimilation until the model has finished: HasFinished() returns true if the simulation is done, false otherwise.

14.1 Extended Kalman filter algorithm

1. Prediction:
   - \( \hat{x}_{h+1} = \mathcal{M}_h(x_h^n) \)
   - \( P_{h+1}^f = M_h P_h^a M_h^T \)
2. Update:

\[-K_{h+1} = P_{f h+1} H_{h+1}^T \left( H_{h+1} P_{f h+1} H_{h+1}^T + R_{h+1} \right)^{-1}\]

\[-x_{a h+1} = x_{h+1} + K_{h+1} \left( y_{h+1} - \mathcal{H}_{h+1}(x_{h+1}^f) \right)\]

\[-P_{a h} = (I - K_{h+1} H_{h+1}) P_{f h+1}\]

With:

- $x_{h}^f$ forecast state vector;
- $x_{h}^a$ analysis state vector;
- $y_{h}$ observation vector;
- $\mathcal{H}_{h}$ observation operator that maps the state space to the observation space;
- $H_{h}$ observation operator linearized at $x_{h}^f$;
- $P_{h}^f$ error covariance matrix of $x_{h}^f$;
- $P_{h}^a$ error covariance matrix of $x_{h}^a$;
- $R_{h}$ observational error covariance matrix;
- $K_{h}$ analysis gain matrix;
- $\mathcal{M}_{h}$ model;
- $M_{h}$ model linearized at $x_{h}^f$. 
Chapter 15

Four Dimensional Variational

Verdandi provides a C++ implementation of the Four Dimensional Variational assimilation method (4DVAR).

The Four Dimensional Variational assimilation method (4DVAR) is implemented in FourDimensionalVariational.hxx and FourDimensionalVariational.cxx. The class FourDimensionalVariational is a template class: FourDimensionalVariational<T, ClassModel, ClassObservationManager, ClassOptimization>. T is the type of the elements to be stored (e.g. double), ClassModel is the type of the model (e.g. ClampedBar<double>), ClassObservationManager is the type of the observation manager (e.g. LinearObservationManager<double>), - ClassOptimization is the type of the optimization algorithm (e.g. Seldon::NLoptSolver).

The implementation of the Four Dimensional Variational assimilation method (4DVAR) relies on the Seldon Optimization Solver to carry out nonlinear optimization. One optimization library is currently interfaced: NLopt.

The installation procedure of the NLopt solver is detailed in the section NLopt Installation.

A simulation with the Four Dimensional Variational assimilation method (4DVAR) may be carried out with the following C++ lines:

FourDimensionalVariational<real, ClampedBar<real>, LinearObservationManager<real>, Seldon::NLoptSolver> driver; // 1

driver.Initialize(argv[1]); // 2

driver.Analyze(); // 3

while (!driver.HasFinished()) // 6
{
    driver.InitializeStep(); // 4

    driver.Forward(); // 5
}

1. First build the FourDimensionalVariational driver with the construction FourDimensionalVariational.

2. Then initialize the driver, the model and the observation manager, and read option keys in the configuration file with Initialize(configuration_file).

3. Compute the initial condition with Analyze(), whenever observations are available.

4. Optionally intialize a step with InitializeStep(). This initializes a step for the model.

5. Perform a step forward and propagate the state error variance with Forward().

6. Compute the data assimilation until the model has finished: HasFinished() returns true if the simulation is done, false otherwise.
15.1 Four Dimensional Variational algorithm

Let consider \((x_h(\xi))_h\) such that \(x_{h=0}(\xi) = x_0 + \xi\).

Given the observations \(y_h\) and the a priori on the initial condition \(x_0\), we want to find the parameter \(\xi\) optimal for the quadratic criterion

\[
J(\xi) = \frac{1}{2} \left\| \xi \right\|^2_{P_0^{-1}} + \frac{1}{2} \sum_{h=0}^{N_t} \left\| y_h - H_h x_h(\xi) \right\|^2_{R_h^{-1}}.
\]

The Four Dimensional Variational assimilation method (4DVAR) carries out the minimization of the previous cost function.

15.1.1 Gradient-based optimization

Gradient descent algorithms are usually used in the minimization of the cost function. We therefore need to compute the gradient of the cost function \(J\).

\(J\) admits a minimum at a point where \(d_\xi J = 0\). Yet,

\[
d_\xi J \cdot \delta \xi = \xi^T P_0^{-1} \delta \xi - \sum_{h=0}^{N_t} (y_h - H_h x_h)^T R_h^{-1} H_h d_\xi x_h \cdot \delta \xi,
\]

with

\[
d_\xi x_{h+1} = M_{h+1} d_\xi x_h,
\]

\[
d_\xi x_0 = 1
\]

Then, we introduce \((p_h)_{1 \leq h \leq N_t}\) fulfilling the adjoint dynamic of \(x_h\),

\[
\begin{cases}
p_h - M_{h+1}^T p_{h+1} = H_h^T R_h^{-1} (y_h - H_h x_h) \\
p_{N_t} = 0
\end{cases}
\]

And we can prove that

\[
d J \cdot \delta \xi = \xi^T P_0^{-1} \delta \xi - p_0^T \delta \xi.
\]

15.1.2 Derivative-free optimization

One can also try derivative-free algorithms such as COBYLA (Constrained Optimization BY Linear Approximations) or PRAXIS (optimization via the "principal-axis method").

15.1.3 Notation

- \(x_h\) state vector;
- \(y_h\) observation vector;
- \(\mathcal{H}_h\) observation operator that maps the state space to the observation space;
- \(H_h\) observation operator linearized at \(x_h\);
- \(P_h\) error covariance matrix of \(x_h\);
- \(R_h\) observational error covariance matrix;
- \(\mathcal{H}_h\) model.
Chapter 16

Monte Carlo

Verdandi provides a C++ implementation of the Monte Carlo method.

The Monte Carlo method is implemented in MonteCarlo.hxx and MonteCarlo.cxx. The class Monte Carlo is a template class: MonteCarlo<T, ClassModel>. T is the type of the elements to be stored (e.g., double), ClassModel is the type of the model (e.g., QuadraticModel<double>).

A simulation with the Monte Carlo method may be carried out with the following C++ lines:

MonteCarlo<real, QuadraticModel<real> > driver; [1]

driver.Initialize(argv[1]); [2]

while (!driver.HasFinished()) [5]
{
    driver.InitializeStep(); [3]
    driver.Forward(); [4]
}

1. First build the Monte Carlo driver with the construction MonteCarlo.

2. Then initialize the driver, the model and the perturbation manager, and read option keys in the configuration file with the method Initialize(configuration_file). Model parameters are perturbed at that stage.

3. Optionally initialize the model before a time step with the method MonteCarlo(). Perturbations are applied to model parameters.

4. Perform a step forward with the method Forward().

5. Compute until the model has finished: the method HasFinished() returns true if the simulation is done, false otherwise.

Note that one call to the Monte Carlo driver will only run a single simulation. In order to compute a full Monte Carlo simulation, the program above should be called as many times as needed.

16.1 Monte Carlo algorithm

The Monte Carlo method consists in perturbing selected parameters from the model. The perturbation is generated randomly and can be applied either at each step or only at the initial time.

1. At time $t_0$, the parameters (or some parameters) $p_0$ of the model are perturbed and denoted $\tilde{p}_0$. The initial condition $x_0$ can also be perturbed.

2. For every time $t_h$, $h \geq 0$,
- $x_{h+1}^f = \mathcal{M}_h(x_h^f, \tilde{p}_h)$,
- the parameters $p_{h+1}$ are perturbed and denoted $\tilde{p}_{h+1}$.

With:
- $x_h^f$ state vector;
- $\mathcal{M}_h$ model;
- $p_h$ model parameters;
- $\tilde{p}_h$ perturbed model parameters.
Chapter 17

Optimal Interpolation

Verdandi provides a C++ implementation of the optimal interpolation.

The optimal interpolation is a sequential data assimilation method. It is implemented in OptimalInterpolation.hxx and OptimalInterpolation.cxx. The class OptimalInterpolation is a template class: OptimalInterpolation<T, ClassModel, ClassObservationManager>. T is the type of the elements to be stored (e.g. double), ClassModel is the type of the model (e.g. ShallowWater<double>), ClassObservationManager is the type of the observation manager (e.g. GridToNetworkObservationManager<double>).

A simulation with the optimal interpolation may be carried out with the following C++ lines:

```cpp
OptimalInterpolation<double, ShallowWater<double>, GridToNetworkObservationManager<double> > driver; [1]
driver.Initialize(argv[1]); [2]
while (!driver.HasFinished()) [6]
{
    driver.InitializeStep(); [3]
    driver.Forward(); [4]
    driver.Analyze(); [5]
}
```

1. First build the OptimalInterpolation driver with the construction OptimalInterpolation.
2. Then initialize the driver, the model and the observation manager, and read the option keys in the configuration file with the method Initialize(configuration_file). This optionally computes an analysis (BLUE) with the model initial condition.
3. Optionally initialize a step for the optimal interpolation with the method InitializeStep(). This initializes a step for the model.
4. Perform a step forward without optimal interpolation with the method Forward().
5. Compute the analysis with the method Analyze(). Whenever observations are available, it assimilates them through the computation of BLUE.
6. Compute the data assimilation until the model has finished: the method HasFinished() returns true if the simulation is done, false otherwise.

17.1 Optimal interpolation algorithm

The optimal interpolation method is a sequential data assimilation method implementing the BLUE (Best Linear Unbiased Estimator) analysis. The BLUE is so-called since (1) it is linearly deduced from the state vector and the observation vector, (2) its error is unbiased and (3) it is optimal in the sense that it has the lowest total variance.
Each time observations are available, BLUE is computed, with prescribed state error variance $B$ and observational error variance $R$. Here is the algorithm:

1. At time $t_0$, the initial condition $x^f_0$ is available.

2. For every time $t_h, h \geq 0$,
   - if observations $y_h$ are available, the analysis takes the BLUE value
     $$x^a_h = x^f_h + K_h(y_h - H_h(x^f_h)),$$
     with
     $$K_h = B_h H^T_h (H_h B_h H^T_h + R_h)^{-1},$$
     and the forecast is
     $$x^f_{h+1} = M_h(x^a_h);$$
   - if there are no observations, the forecast is
     $$x^f_{h+1} = M_h(x^f_h).$$

With:
- $x^f_h$ background state vector;
- $x^a_h$ analysis state vector;
- $y_h$ observation vector;
- $H_h$ linear observation operator that maps the state space to the observation space;
- $H_h$ observation operator linearized at $x^f_h$;
- $B_h$ background error covariance matrix (error variance of $x^f_h$);
- $R_h$ observational error covariance matrix;
- $K_h$ analysis gain matrix;
- $M_h$ model.
Chapter 18

Reduced Minimax Filter

Verdandi provides a C++ implementation of the Reduced Minimax Filter (RMF).

The Reduced Minimax Filter (RMF) is implemented in ReducedMinimax.hxx and ReducedMinimax.cxx. The class ReducedMinimax is a template class: ReducedMinimax<T, ClassModel, ClassObservationManager>. T is the type of the elements to be stored (e.g., double), ClassModel is the type of the model (e.g., ClampedBar<double>), ClassObservationManager is the type of the observation manager (e.g., LinearObservationManager<double>).

A simulation with the Reduced Minimax Filter (RMF) may be carried out with the following C++ lines:

```cpp
ReducedMinimax<real, ClampedBar<real>,
LinearObservationManager<real> > driver; 
```

1. First build the ReducedMinimax driver with the construction ReducedMinimax.

2. Then initialize the driver, the model and the observation manager, and read option keys in the configuration file with Initialize(configuration_file).

3. Initialize a step with InitializeStep(). This initializes a step for the model.

4. Perform a step forward, with assimilation of the observation, and propagate the minimax gain with Forward().

5. Iterate until the model has finished: HasFinished() returns true if the simulation is done, false otherwise.

18.1 Reduced Minimax filter algorithm

1. Initialization:
   - \( \hat{x}_0 = G_0^{-1} \left( F_0^T S_0^{-1} \overline{\pi} + H_0^T R_0^{-1} (y_0 - \overline{\eta}_0) \right) \)
   - \( G_0 = F_0^T S_0^{-1} F_0 + H_0^T R_0^{-1} H_0 \)
   - \( \beta_0 = \langle R_0^{-1} (y_0 - \overline{\eta}_0), y_0 - \overline{\eta}_0 \rangle \)

2. Update:
   - \( x_{b+1}^f = \mathcal{M}_b(F_b \hat{x}_b) \)
- $\hat{x}_{h+1} = F_{h+1}^T x_{h+1}^f + G_{h+1}^{-1} H_{h+1}^T R_{h+1}^{-1} [y_{h+1} - \overline{\eta}_{h+1}] - H_{h+1} F_{h+1}^T x_{h+1}^f + c_{h+1},$
- $c_{h+1} = G_{h+1}^{-1} F_{h+1}^T (Q_{h} + M_h B_h M_h^T Q_{h}^{-1}) x_{h+1}^f + \overline{w}_{h},$
- $G_{h+1} = F_{h+1}^T [Q_{h}^{-1} - Q_{h}^{-1} M_h B_h M_h^T Q_{h}^{-1}] F_{h+1} + H_{h+1}^T R_{h+1}^{-1} H_{h+1},$
- $B_{h} = (G_{h} + M_h Q_{h}^{-1} M_h)^{-1}$
- $\beta_{h+1} = \beta_{h} - <B_{h+1}^{-1} \hat{x}_{h}, \hat{x}_{h}> + <F_{h} M_{h}^T Q_{h} M_{h} F_{h}^T \overline{w}_{h}, \overline{w}_{h}> + \gamma_{h+1}$
- $\gamma_{h+1} = <R_{h}^{-1} (y_{h} - \overline{\eta}_{h}), y_{h} - \overline{\eta}_{h}>$

3. Reachability set:

- $\mathcal{R}(h) = \hat{x}_{h} + \sqrt{1 - \beta_{h}} + <G_{h} \hat{x}_{h}, \hat{x}_{h}> \mathcal{R}(t)$
- $\mathcal{R}(h) = \{z : <G_{h} z, z> \leq 1\}$

With:

- $x_{h}^f$ forecast state vector;
- $\hat{x}_{h}$ analysis state vector;
- $y_{h}$ observation vector;
- $H_{h}$ observation operator that maps the state space to the observation space;
- $G_{h}$ minimax gain matrix;
- $R_{h}$ observational error covariance matrix, possibly scaled;
- $Q_{h}$ state error covariance matrix, possibly scaled;
- $\mathcal{R}(h)$ a set describing how the model propagates uncertain initial condition, observation error and model error;
- $\beta_{h}$ observation-dependent scaling factor;
- $M_{h}$ model;
- $\tau$ systematic error in the initial condition;
- $\overline{\tau}_{h}$ systematic model error;
- $\overline{\eta}_{h}$ systematic observation error;
- $M_{h}$ tangent linear model linearized at $\hat{x}_{h}$;
- $F_{h}^T$ is a reduction matrix mapping the state space into the reduced-state space.
Chapter 19

Reduced Order Extended Kalman Filter

Verdandi provides a C++ implementation of the reduced order extended Kalman filter (ROEKF) also known as singular evolutive extended Kalman filter (SEEK).

The reduced order extended Kalman filter (ROEKF) is implemented in ReducedOrderExtendedKalmanFilter.hxx and ReducedOrderExtendedKalmanFilter.cxx. The class ReducedOrderExtendedKalmanFilter is a template class: ReducedOrderExtendedKalmanFilter<T, ClassModel, ClassObservationManager>. T is the type of the elements to be stored (e.g. double), ClassModel is the type of the model (e.g. ClampedBar<double>), ClassObservationManager is the type of the observation manager (e.g. LinearObservationManager<double>).

A simulation with the reduced order extended Kalman filter (ROEKF) may be carried out with the following C++ lines:

```cpp
ReducedOrderExtendedKalmanFilter<real, ClampedBar<real>, LinearObservationManager<real>> driver; // [1]
driver.Initialize(argv[1]); // [2]
while (!driver.HasFinished()) { // [6]
    driver.InitializeStep(); // [3]
    driver.Forward(); // [4]
    driver.Analyze(); // [5]
}
```

1. First build the ReducedOrderExtendedKalmanFilter driver with the construction ReducedOrderExtendedKalmanFilter.
2. Then initialize the driver, the model and the observation manager, and read option keys in the configuration file with Initialize(configuration_file). This optionally computes an analysis with the model initial condition.
3. Optionally initialize a step with InitializeStep(). This initializes a step for the model.
4. Perform a step forward and propagate the state error variance with Forward().
5. Compute the analysis with Analyze(), whenever observations are available.
6. Compute the data assimilation until the model has finished: HasFinished() returns true if the simulation is done, false otherwise.
19.1 Reduced Order Extended Kalman filter algorithm

Assuming that $P$ is of reduced rank $p$ – typically much smaller than the dimension of the space $n$ – the basic idea in reduced-order filtering is, in essence, to be able to manipulate covariance matrices in the factorized form

$$P = LU^{-1}L^T,$$

where $U$ – in the group of invertible matrices $GL_p$ – is of much smaller size than $P \in M_n$ and represents the main uncertainties in the system. What is crucial here is to be able to perform all computations on $L$ and $U$ without needing to compute $P$.

1. Prediction:

   $x_{h+1}^f = \mathcal{M}_h(x_h^a) + Q_{h+1}$

2. Update:

   $L_{h+1} = M_h L_h$
   $U_{h+1} = U_h + (H_{h+1} L_{h+1})^T R_{h+1}^{-1} H_{h+1} L_{h+1}$
   $x_{h+1}^a = x_{h+1}^f + L_{h+1} U_{h+1}^{-1} (H_{h+1} L_{h+1})^T R_{h+1}^{-1} (y_{h+1} - H_{h+1} x_{h+1}^f)$

With:

- $x_h^f$ forecast state vector;
- $x_h^a$ analysis state vector;
- $y_h$ observation vector;
- $\mathcal{M}_h$ observation operator that maps the state space to the observation space;
- $H_h$ observation operator linearized at $x_h^f$;
- $Q_h$ model error covariance matrix;
- $R_h$ observational error covariance matrix;
- $\mathcal{M}_h$ model.
Chapter 20

Reduced Order Unscented Kalman Filter

Verdandi provides a C++ implementation of the reduced order unscented Kalman filter (ROUKF), which is the generalized formulation of the singular evolutive interpolated Kalman filter (SEIK).

The reduced order unscented Kalman filter (ROUKF) is implemented in ReducedOrderUnscentedKalmanFilter.hxx and ReducedOrderUnscentedKalmanFilter.cxx. The class ReducedOrderUnscentedKalmanFilter is a template class: ReducedOrderUnscentedKalmanFilter<T, ClassModel, ClassObservationManager>. T is the type of the elements to be stored (e.g. double), ClassModel is the type of the model (e.g. ClampedBar<double>), ClassObservationManager is the type of the observation manager (e.g. LinearObservationManager<double>).

A simulation with the reduced order unscented Kalman filter (ROUKF) may be carried out with the following C++ lines:

```cpp
ReducedOrderUnscentedKalmanFilter<real, ClampedBar<real>,
    LinearObservationManager<real>> driver; [1]
driver.Initialize(argv[1]); [2]
while (!driver.HasFinished()) [6]
{
    driver.InitializeStep(); [3]
    driver.Forward(); [4]
    driver.Analyze(); [5]
}
```

1. First build the ReducedOrderUnscentedKalmanFilter driver with the construction ReducedOrderUnscentedKalmanFilter.
2. Then initialize the driver, the model and the observation manager, and read option keys in the configuration file with Initialize(configuration_file). This optionally computes an analysis with the model initial condition.
3. Optionally initialize a step with InitializeStep(). This initializes a step for the model.
4. Perform a step forward and propagate the state error variance with Forward().
5. Compute the analysis with Analyze(), whenever observations are available.
6. Compute the data assimilation until the model has finished: HasFinished() returns true if the simulation is done, false otherwise.
20.1 Reduced Order Unscented Kalman filter algorithm

Assuming that \( P \) is of reduced rank \( p \) – typically much smaller than the dimension of the space \( n \) – the basic idea in reduced-order filtering is, in essence, to be able to manipulate covariance matrices in the factorized form

\[
P = LU^{-1}L^T,
\]

where \( U \) – in the group of invertible matrices \( \mathcal{G} \mathcal{L}_p \) – is of much smaller size than \( P \in \mathcal{M}_n \) and represents the main uncertainties in the system. What is crucial here is to be able to perform all computations on \( L \) and \( U \) without needing to compute \( P \).

20.2 Simplex case

In this section, we focus on the simplex distribution. Consider some simplex sigma-points \( (V^{(i)})_{1 \leq i \leq r} \in \mathbb{R}^p \) associated with some coefficients \( (\alpha) = (\alpha_1...\alpha_r)^T \). Then, we define the matrix of these sigma-points denoted by \([V^*] \in \mathcal{M}_{r,p}\) and the matrix \( D_\alpha = \text{diag}(\alpha_1...\alpha_r) \in \mathcal{M}_r \).

1. Sampling:
   - \( C_h = \sqrt{U_h^{-1}} \),
   - \( x_h^{(i)a} = x_h^a + L_h C_h f^{(i)}, \quad 1 \leq i \leq p + 1 \)

2. Prediction:
   - \( x_{h+1}^{(i)f} = E_\alpha(\mathcal{H}(x_{h+1}^{(i)a})) \)
   - \( x_{h+1}^{(i)f} = \begin{cases} x_{h+1}^{(i)+} [\mathcal{H}(x_{h+1}^{(i)a})] D_\alpha [V^*]^T ([V^*][D_\alpha[V^*]^T]^{-1/2})^{(i)} & \text{with resampling} \\ \mathcal{H}(x_{h+1}^{(i)a}) & \text{without resampling} \end{cases} \)
   - \( L_{h+1} = [x_{h+1}^{(i)f}] D_\alpha [V^*]^T \in \mathcal{M}_{n,p} \)
   - \( P_{h+1}^f = L_{h+1} (P_{\alpha}^a)^{-1} L_{h+1}^T \)

3. Update:
   - \( y_{h+1}^{(i)} = \mathcal{H}_{h+1}(x_{h+1}^{(i)f}) \)
   - \( [HL]_{h+1} = [y_{h+1}^a] D_\alpha [V^*]^T \)
   - \( U_{h+1} = I + [HL]_{h+1} R_{h+1}^{-1} [HL]_{h+1} \in \mathcal{M}_p \)
   - \( x_{h+1}^a = x_{h+1}^{(i)+} + L_{h+1} U_{h+1}^{-1} [HL]_{h+1} R_{h+1}^{-1} (y_{h+1} - E_\alpha(y_{h+1}^{(i)})) \)
   - \( P_{h+1}^a = L_{h+1} U_{h+1}^{-1} L_{h+1}^T \)

With:
- \( x_h^f \) forecast state vector;
- \( x_h^a \) analysis state vector;
- \( y_h \) observation vector;
- \( \mathcal{H}_h \) observation operator that maps the state space to the observation space;
- \( H_h \) observation operator linearized at \( x_h^f \);
- \( P_h^f \) error covariance matrix of \( x_h^f \);
- \( P_h^a \) error covariance matrix of \( x_h^a \);
- \( R_h \) observational error covariance matrix;
Given adequate sampling rules, precompute the corresponding \( [V^*], P^V_\alpha = [V^*]D_\alpha[V^*]^T, [P^*] = ([V^*]D_\alpha[V^*]^T)^{-\frac{1}{2}}[V^*], \) and \( D_V = D_\alpha[V^*]^T(P^V_\alpha)^{-1}[V^*]D_\alpha. \)

1. **Sampling:**
   
   \[- C_h = \sqrt{U_h^{-1}} \]
   \[- x_h^{(i)n} = x_h^n + L_hC_hf(t), \quad 1 \leq i \leq p + 1 \]

2. **Prediction:**
   
   \[- x_h^{(i)f} = E_a(\mathcal{H}_h(x_h^{(i)+})) \]
   \[- x_h^{(i)f} = x_{h+1}^f + [\mathcal{H}_h(x_h^a) - x_{h+1}^f]D_\alpha^{1/2}Y_pf(t), \text{ resampling with SVD} \]
   \[- L_h^{f+1} = [x_h^{(i)f}]D_\alpha[V^*]^T \in \mathcal{M}_{n,p} \]
   \[- p_h^{(i)f} = L_h^{f+1}(p^V_\alpha)^{-1}L_h^{-1} \]

3. **Update:**
   
   \[- [\tilde{y}] = [\mathcal{H}_h(x_h^{(i)+})] - E_a(\mathcal{H}_h(x_h^{(i)f})) \]
   \[- D_m = [\tilde{y}]R_{h+1}^{-1}[\tilde{y}] \in \mathcal{M} \]
   \[- U_{h+1} = P_h^\alpha + [V^*]D_\alpha(1 + D_m(D_\alpha - D_V))^{-1}D_mD_\alpha[V^*]^T \in \mathcal{M}_p \]
   \[- [HL]_{h+1} = [\tilde{y}]\left(1 + D_m(D_\alpha - D_V))^{-1}\right)^{-1}D_mD_\alpha[V^*]^T \]
   \[- x_h^{(i)+} = x_h^{(i)f} + L_{h+1}U_{h+1}^{-1}[HL]_{h+1}^{-1}R_{h+1}^{-1}(y_{h+1} - E_a(x_h^{(i)+})) \]
   \[- p_h^{(i)+} = L_{h+1}U_{h+1}^{-1}L_{h+1}^{-1} \]

With:

- \( x_h^f \) forecast state vector;
- \( x_h^a \) analysis state vector;
- \( y_h \) observation vector;
- \( \mathcal{H}_h \) observation operator that maps the state space to the observation space;
- \( H_h \) observation operator linearized at \( x_h^f \);
- \( p_h^f \) error covariance matrix of \( x_h^f \);
- \( p_h^a \) error covariance matrix of \( x_h^a \);
- \( R_h \) observational error covariance matrix;
- \( \mathcal{M}_h \) model.

## 20.4 Reference

For more detail about the reduced order unscented Kalman filtering see

*Reduced-order Unscented Kalman Filtering with application to parameter identification in large-dimensional systems* (P. Moireau, D. Chapelle).
Chapter 21

Unscented Kalman Filter

Verdandi provides a C++ implementation of the unscented Kalman filter (UKF).

The unscented Kalman filter (UKF) is implemented in UnscentedKalmanFilter.hxx and UnscentedKalmanFilter.hxx. The class UnscentedKalmanFilter is a template class: UnscentedKalmanFilter<T, ClassModel, ClassObservationManager>. T is the type of the elements to be stored (e.g. double), ClassModel is the type of the model (e.g. ClampedBar<double>), and ClassObservationManager is the type of the observation manager (e.g. LinearObservationManager<double>).

A simulation with the unscented Kalman filter (UKF) may be carried out with the following C++ lines:

```cpp
UnscentedKalmanFilter<real, ClampedBar<real>, LinearObservationManager<real>> driver; [1]
driver.Initialize(argv[1]); [2]
while (!driver.HasFinished()) [6] 
{
    driver.InitializeStep(); [3]
    driver.Forward(); [4]
    driver.Analyze(); [5]
}
```

1. First build the UnscentedKalmanFilter driver with the construction UnscentedKalmanFilter.
2. Then initialize the driver, the model and the observation manager, and read option keys in the configuration file with Initialize(configuration_file). This optionally computes an analysis with the model initial condition.
3. Optionally initialize a step with InitializeStep(). This initializes a step for the model.
4. Perform a step forward and propagate the state error variance with Forward().
5. Compute the analysis with Analyze(), whenever observations are available.
6. Compute the data assimilation until the model has finished: HasFinished() returns true if the simulation is done, false otherwise.

21.1 Unscented Kalman filter algorithm

The unscented Kalman filter (UKF) is based on using well-chosen “interpolation points” (sigma-points) in order to propagate the mean and covariance of a random variable with improved accuracy with respect to standard extended Kalman filtering (EKF). Different choices of sigma-points are implemented in Verdandi:
1. canonical sigma-points: aligned with the canonical \((e_i)\) of the space with associated coefficient \(\alpha_i = \frac{1}{2n}\)
   - \(I^{(i)} = \sqrt{\alpha_i} e_i\), for \(1 \leq i \leq n\)
   - \(I^{(i)} = -\sqrt{\alpha_i} e_i\), for \(n + 1 \leq i \leq 2n\);
2. star sigma-points: the origin is added to the previous canonical points;
3. simplex sigma-points: this represents the smallest number of necessary sigma-points \(r = n + 1\), which are located on a regular polyhedron of radius \(\sqrt{n}\).

The principle of the UKF filter is to replace the means and covariances of the Kalman filter by the empirical means and covariances propagated by the dynamical operator \(\mathcal{M}\) during the prediction, and by the observation operator \(\mathcal{H}\) during the correction. This lead the following algorithm:

1. Prediction:
   - \(x^{(i)a}_h = x^a_h + \sqrt{P^a_h I^{(i)}}\)
   - \(x^{(i)}_{h+1} = E\alpha(\mathcal{M}_h(x^{(i)a}_h))\)
   - \(P^f_{h+1} = \text{Cov}_a(\mathcal{M}_h(x^{(i)a}_h))\)

2. Update:
   - \(x^{(i)f}_{h+1} = x^f_{h+1} + \sqrt{P^f_{h+1} I^{(i)}}\)
   - \(y^{(i)}_{h+1} = \mathcal{H}_h(x^{(i)f}_{h+1})\)
   - \(P^a_{a} = \text{Cov}_a(x^{(i)}_{h+1}, y^{(i)}_{h+1})\)
   - \(P^a_{a} = R_{h+1} = \text{Cov}_a(y^{(i)}_{h+1}, y^{(i)}_{h+1})\)
   - \(K_{h+1} = P^a_{a} (P^a_{a})^{-1}\)
   - \(x^a_{h+1} = x^f_{h+1} + K_{h+1} (y_{h+1} - E\alpha(y^{(i)}_{h+1}))\)
   - \(P^a_{h+1} = P^f_{h+1} - P^a_{a} (P^a_{a})^{-1} (P^a_{a})^{T}\)

With:
- \(x^f_h\) forecast state vector;
- \(x^a_h\) analysis state vector;
- \(y_h\) observation vector;
- \(\mathcal{H}_h\) observation operator that maps the state space to the observation space;
- \(H_h\) observation operator linearized at \(x^f_h\);
- \(P^f_h\) error covariance matrix of \(x^f_h\);
- \(P^a_h\) error covariance matrix of \(x^a_h\);
- \(R_h\) observational error covariance matrix;
- \(K_h\) analysis gain matrix;
- \(\mathcal{M}_h\) model.

### 21.2 Reference

For more detail about the unscented Kalman filtering see

Reduced-order Unscented Kalman Filtering with application to parameter identification in large-dimensional systems (P. Moireau, D. Chapelle).
Part IV

Parallelism
Chapter 22

Parallelism in Verdandi

Verdandi provides two levels of parallelization: some data assimilation methods can instantiate several models in parallel, each of this model's instance could be itself parallelized. The data assimilation methods are parallelized by MPI and only models parallelized by MPI are yet supported. MPI has been chosen for its portability and its performance capabilities in both shared-memory multiprocessors (massively parallel machines) and distributed-memory multiprocessors (heterogeneous cluster).

22.1 Parallel Method applied to Sequential Model

This section describes the parallel data assimilation methods that can be applied to sequential models. The Section 22.1.1 introduced the parallelization of the algorithms 'ReducedOrderExtendedKalmanFilter' [13] and 'ReducedOrderUnscentedKalmanFilter'. The Section 22.1.2 explains the use of these data assimilation methods applied to the sequential example model 'ClampedBar'. The performance of these algorithms are presented in Section 22.1.3.

22.1.1 Parallel Algorithms

22.1.1.0.4 Parallelization of the 'ReducedOrderExtendedKalmanFilter'

Algorithm

1. Prediction:
   \[ x_{h+1}^f = \mathcal{M}_h(x_{h}^a) \]

2. Update:
   \[ L_{h+1} = M_h L_h \]
   \[ U_{h+1} = U_h + (H_{h+1} L_{h+1})^T R_{h+1}^{-1} H_{h+1} L_{h+1} \]
   \[ x_{h+1}^a = x_{h+1}^f + L_{h+1} U_{h+1}^{-1} (H_{h+1} L_{h+1})^T R_{h+1}^{-1} (y_{h+1} - H_{h+1} x_{h+1}^f) \]

With:
- \( x_h^f \) forecast state vector;
- \( x_h^a \) analysis state vector;
- \( y_h \) observation vector;
- \( \mathcal{M}_h \) observation operator that maps the state space to the observation space;
- \( H_h \) observation operator linearized at \( x_h^f \);
- \( Q_h \) model error covariance matrix;
$R_h$ observational error covariance matrix;
$\mathcal{M}_h$ model.

**Parallelization of the $L$ computation**

We describe, in this part, the parallelization of the sensitivity matrix update:

$$L_{h+1} = M_h L_h$$

During a simulation, every process used has its own instance of model. The columns of the matrix $L$ are distributed in equal amounts to all the processes (see Figure 22.1.0.4). The tangent model is applied in parallel on each column of the local sub matrix $L_p$.

![Figure 22.1: Distribution of the matrix $L$ into local sub-matrices $L_p$](image)

**Parallelization of the $U$ computation**

We focus in this part on the computation of the reduced covariance matrix $U$:

$$U_{h+1} = U_h + (H_{h+1} L_{h+1})^T R_{h+1}^{-1} H_{h+1} L_{h+1}$$

- The tangent observation operator is applied in parallel on each column of the local sub matrix $L_p$ (see Figure 22.2). The resulted matrix $H L_p$ is a sub-matrix of $HL$. The columns of $HL_p$ correspond to the same column indices as those available of matrix $HL_p$.

- Each process sends its local sub-matrix $HL_p$ to all others (allgather operation). Thus, each process owns the entire matrix $HL$.

- Each process computes the sub-matrix $(R_{h+1}^{-1} H_{h+1} L)_{p}$ of $R_{h+1}^{-1} H_{h+1} L_{h+1}$:
  $$ (R_{h+1}^{-1} H_{h+1} L)_{p} = R_{h+1}^{-1} (H_{h+1} L_{h+1})_{p}$$  (see Figure 22.2)

- Each process computes the sub-matrix $(U_{h+1})_{p}$ of the reduced covariance matrix $U_{h+1}$:
  $$ (U_{h+1})_{p} = (U_h)_{p} + (H_{h+1} L_{h+1})^T (R_{h+1}^{-1} H_{h+1} L_{h+1})_{p}$$  (see Figure 22.2)

$^1 L_p$ is the sub-matrix of $L$ available on the process of rank $p$. 
22.1 Parallel Method applied to Sequential Model

![Matrix-matrix product type 1](image)

**Figure 22.2: Matrix-matrix product type 1**

- Each process sends its local sub-matrix \((U_{h+1})_p\) to all others (allgather operation). Thus, each process owns the entire matrix \(U_{h+1}\).

Parallelization of the \(x^a\) computation

Below is explained the update of the model state vector:

\[
x^a_{h+1} = x^f_{h+1} + L_{h+1} U_{h+1}^{-1} (H_{h+1} L_{h+1})^T R_{h+1}^{-1} (y_{h+1} - H_{h+1} x^f_{h+1})
\]

- Each process computes the innovation: \(z = (y_{h+1} - H_{h+1} x^f_{h+1})\).

- The computation \(d_p = (R_{h+1}^{-1} H_{h+1} L_{h+1})^T z\) is performed in parallel by each process. Only \(k_p\) rows of matrix \((R^{-1} HL)^T\) are available locally (transpose of a matrix distributed by column). The innovation vector \(z\) is fully allocated on each process. Consequently, each process is able to compute locally the \(k_p\) elements of the vector \(d\) whose indices correspond to those of the row of \((R^{-1} HL)^T\) available locally (see 22.3).

- Each process sends its local sub-vector \(d_p\) to all others (allgather operation). Thus, each process owns the entire vector \(d\).

- The system \(U_{h+1} c = d\) is solved by each process.

- Each process computes the local contribution \(\Delta x_p = (L_{h+1})_p c\):

  Only \(k_p\) columns of matrix \(L_{h+1}\) and \(k_p\) rows of vector \(c\) are available locally. The product \(\Delta x_p = (L_{h+1})_p c\) is a vector of the same size of \(x^a\) which elements represent a partial sum of the product \(L_{h+1} c\) (see Figure 22.4). Thus, to obtain the product \(L_{h+1} c\), the contribution of all the processes have to be summed (all reduce operation).

22.1.1.0.5 Parallelization of the ‘ReducedOrderUnscentedKalmanFilter’

Algorithm

1. **Sampling:**
   - \(C_h = \sqrt{U_h^{-1}}\)
   - \(x^{(i)a}_h = x^a_h + L_h C_h f^{(i)}, \quad 1 \leq i \leq p + 1\)
\[(\begin{bmatrix} R^1 H L \end{bmatrix})^T \begin{bmatrix} z \end{bmatrix} = \begin{bmatrix} d_p \end{bmatrix}\]

Figure 22.3: Matrix-matrix product type 2

\[(\begin{bmatrix} L_p \end{bmatrix}) \begin{bmatrix} d \end{bmatrix} = \begin{bmatrix} \Delta x_p \end{bmatrix}\]

Figure 22.4: Matrix-matrix product type 3
2. Prediction:

- \( x_{h+1}^{f} = E_{\alpha}(\mathcal{K}_{h}(x_{h+1}^{(s)})) \)
- \( x_{h+1}^{(i)} = x_{h+1}^{f} + [\mathcal{K}_{h}(x_{h}^{a}) - x_{h+1}^{f}][D_{\alpha}^{1/2}Y_{\mu}]^{T} \), resampling with SVD
- \( L_{h+1} = [x_{h+1}^{(i)}]D_{\alpha}[V^{*}]^{T} \in \mathcal{M}_{n,p} \)
- \( P_{h+1}^{f} = L_{h+1}(P_{\alpha}^{f})^{-1}L_{h+1}^{T} \)

3. Update:

- \([\tilde{y}] = [\mathcal{K}_{h+1}(x_{h+1}^{(s)}) - E_{\alpha}(\mathcal{K}_{h+1}(x_{h+1}^{(s)}))]\)
- \( D_{m} = [\tilde{y}]^{T}R_{h+1}^{-1}[\tilde{y}] \in \mathcal{M}_{r} \)
- \( U_{h+1} = P_{\alpha}^{f} + [V^{*}]D_{\alpha}(1 + D_{m}(D_{\alpha} - D_{V}))^{-1}D_{m}D_{\alpha}[V^{*}]^{T} \in \mathcal{M}_{p} \)
- \( \{HL\}_{h+1} = [\tilde{y}](1 + D_{\alpha}D_{m})^{-1}(1 + D_{V}(1 + D_{m}(D_{\alpha} - D_{V}))^{-1}D_{m})D_{\alpha}[V^{*}]^{T} \)
- \( x_{h+1}^{s} = x_{h+1}^{f} + L_{h+1}U_{h+1}^{-1}\{HL\}_{h+1}^{T}R_{h+1}^{-1}(y_{h+1} - E_{\alpha}(y_{h+1})) \)
- \( P_{h+1}^{a} = L_{h+1}U_{h+1}^{-1}L_{h+1}^{T} \)

With:

- \( x_{h}^{f} \) forecast state vector;
- \( x_{h}^{s} \) analysis state vector;
- \( y_{h} \) observation vector;
- \( \mathcal{K}_{h} \) observation operator that maps the state space to the observation space;
- \( H_{h} \) observation operator linearized at \( x_{h}^{f} \);
- \( P_{h}^{f} \) error covariance matrix of \( x_{h}^{f} \);
- \( P_{h}^{a} \) error covariance matrix of \( x_{h}^{a} \);
- \( R_{h} \) observational error covariance matrix;
- \( \mathcal{M}_{h} \) model.

- The particles \( x_{h}^{(i)a} \) are distributed over the processes.
- Each process executes the sequential ROUKF algorithm with its local particles.
- The matrices \( x_{h+1}, L_{h+1}, (HL)_{h+1} \) and \( U_{h+1} \) are updated with the different parallel matrix-matrix products presented in Section 22.1.1.0.4.

22.1.2 Example Programs

The examples are located in the example/clamped_bar directory. ²

22.1.2.0.6 Compilation

First of all, the preprocessor variable \( \text{VERDANDI\_WITH\_MPI} \) has to be defined in files reduced_order_extended_kalman_filter.cpp and reduced_order_unscented_kalman_filter.cpp:

```
#define VERDANDI_DEBUG_LEVEL_4
#define SELDON_WITH_BLAS
#define SELDON_WITH_LAPACK
#define VERDANDI_WITH_ABORT
#define VERDANDI_DENSE
```

²To have a summary of Verdandi contents see Section 8.
Then, compile the program `generate_observation.cpp`:

```
$ scons generate_observation
```

Finally, compile the programs `reduced_order_extended_kalman_filter.cpp` and `reduced_order_unscented_kalman_filter.cpp` with the option 'mpi=yes':

```
$ scons reduced_order_extended_kalman_filter mpi=yes
$ scons reduced_order_unscented_kalman_filter mpi=yes
```

22.1.2.0.7 Observation

Since no observations are given yet, we have to generate some. Execute the following command:

```
host<~/> ./generate_observation configuration/truth.lua
```

to run the model with the initial conditions described in `truth.lua`, without data assimilation. This should generate a result file (`truth-state_forecast.bin`) in the directory `example/clamped_bar/result/`. This file store the state (displacement, velocity, $\theta_f$) trajectory.

The generated state (displacement, velocity, $\theta_f$) will serve as observations for the assimilation.

22.1.2.0.8 Data Assimilation with ROEKF and ROUKF

To use the Reduced Order Extended Kalman Filter and the Reduced Order Unscented Kalman Filter methods, execute the following commands.

```
$ mpirun -n 2 reduced_order_extended_kalman_filter configuration/assimilation.lua
$ mpirun -n 2 reduced_order_unscented_kalman_filter configuration/assimilation.lua
```

This runs the model with the initial conditions described in `example/clamped_bar/configuration/assimilation.lua`. The simulation begins with erroneous values for the parameter $\theta_f$. This should generate the same results as for the sequential simulation.

**Warning:** The number of processes should be less than or equal to the size of the reduced model state vector.

22.1.3 Performance

The figures 22.5 and 22.6 introduce the resulting performance of the ROEKF and ROUKF algorithms applied to the sequential model ClampedBar. These simulations were performed on 2 x 3 GHz Quad-Core Intel Xeon with a 16 GB of memory in which the processes used during simulation were placed at arbitrary cores relative to the process constructing the network.
22.1 Parallel Method applied to Sequential Model

Figure 22.5: Speed up of the parallel ROEKF algorithm applied to the sequential model ClampedBar with $N_{\text{state}} = 10^4$, $N_{\text{observation}} = 10^2$ and $N_{\text{sigma\_point}} = 16$.

Figure 22.6: Speed up of the parallel ROUKF algorithm applied to the sequential model ClampedBar with $N_{\text{state}} = 10^4$, $N_{\text{observation}} = 10^2$ and $N_{\text{sigma\_point}} = 16$. 
Parallelism in Verdandi

22.2 Sequential Method applied to Parallel Model

Verdandi intends to provide the ability to apply its data assimilation methods to models whose state vector is distributed on several processes. In the case of large scale state vector, neither the model nor the data assimilation method can afford to allocate a variable of this size. Consequently, several data assimilation variables could be distributed, for instance the sensitivity matrix and the variance. The number of components to be stored locally has to be compatible with the distributed model state vector for parallel matrix-vector operations.

The management of the types in Verdandi, detailed in Section 22.2.1, enabled to implement this capability easily. The chosen solution was to create an interface between Seldon, the linear algebra library used in Verdandi, and PETSc a framework for parallel computing. This choice do not confine the type of the model state vector only to PETSc distributed structures since any MPI distributed structures can be encapsulated or copied in a PETSc object.

22.2.1 Types Management in Verdandi

The implementation of the Verdandi algorithms relies on the linear algebra library Seldon. This library provides different matrix and vector structures, and many functions for computations (linear algebra). It provides matrices for two main categories: dense matrices and sparse matrices. Among dense matrices, there are specific structures for rectangular matrices, symmetric matrices, hermitian matrices and triangular matrices. Each type includes several formats: rectangular dense matrices may be stored by rows or by columns; symmetric dense matrices may be stored as rectangular matrices or only upper part of the matrix is stored. Many different types for sparse matrix are also available. All of these matrix classes share the same interface. Linear algebra computation functions are template functions and many BLAS operations bringing into play different matrix types are implemented.

Example program which computes the product of a dense matrix by a sparse matrix:

```cpp
// Dense matrix.
Matrix<double, General, RowMajor> A(3, 3), C(3, 3);
A.Fill();
C.Fill();
// Sparse matrix.
Matrix<double, General, ArrayRowSparse> B(3, 3);
B(0, 0) = 2.0;
B(1, 0) = 1.0;
// Computes matrix–matrix product alpha*A*B + beta*C --> C.
MltAdd(1.0, A, B, 2.0, C);
```

In Verdandi, the model and the observation manager can provide their vector and matrix types to the data assimilation method thanks to the C++ ‘typedef’ mechanism. For instance:

```cpp
class Model
{
    public:
    /// Type of the state error variance.
    typedef Matrix<double, General, RowSparse> state_error_variance;
    /// Brief Type of the reduced matrix \(LUL^T\) in the \(LUL^T\) decomposition of the state error variance. */
    typedef Matrix<double, General, RowMajor> state_error_variance_reduced;
    /// Type of the state vector.
    typedef Vector<double> state;
    ...
}

class ObservationManager
{
    public:
    /// Type of the tangent linear operator.
    typedef Matrix<double, General, RowSparse> tangent_linear_operator;
    /// Type of the observation vector.
    typedef Vector<double> observation;
    ...
}
```

Thus, the data assimilation method is able to get back the type of the object to instantiate. For instance, to fetch the type of the model state vector:
Sequential Method applied to Parallel Model

```
template <class Model, class ObservationManager>
class DataAssimilationMethod
{

    // Type of the model state vector.
    typedef typename Model::state model_state;
}
```

### 22.2.2 Distributed Structure in Seldon

To ensure the compatibility of the Verdandi data assimilation methods with distributed models, it was sufficient to:

- add vector and matrix distributed structures to Seldon.
- implement the linear algebra computation functions for these new types, required by the concerned data assimilation methods.

#### 22.2.2.0.9 Distributed Vector

The class `Seldon::Vector<double, PETScPar>` encapsulate a distributed PETSc vector of type `VecMPI`. This class implement the same interface as a classic Seldon vector. The whole set of BLAS1 operations have been implemented for this type.

The source files of the class `Seldon::Vector<double, PETScPar>` are located in the `seldon/vector/` directory. Several methods are specific of `Seldon::Vector<double, PETScPar>` class:

```
template <class T, class Allocator>
class Vector<T, PETScPar, Allocator>:

    public PETScVector<T, Allocator>
{

    // Returns a reference on the inner petsc vector.
    Vec& GetPetscVector();

    // Returns a const reference on the inner petsc vector.
    const Vec& GetPetscVector() const;

    // Sets the MPI communicator.
    void SetCommunicator(MPI_Comm mpi_communicator);

    // Inserts or adds values into certain locations of a vector.
    // Warning These values may be cached, so 'Flush' must be called after
    // all calls to SetBuffer() have been completed.
    void SetBuffer(int i, T value, InsertMode insert_mode = INSERT_VALUES);

    // Assembles the PETSc vector.
    void Flush();

    // Returns the range of indices owned by this processor.
    // The vectors are laid out with the first \$n_1\$ elements on the first
    // processor, next \$n_2\$ elements on the second, etc. If the current
    // processor is \$k\$, this method returns \$n_k\$ in \$a\$ and
    // \$n_{k+1}\$ in \$b\$. If \$a\$ is set to PETSC_NULL on entry, it is not
    // modified by this function. Same is true for \$b\$.
    void GetProcessorRange(int& i, int& j) const;

    ...
}
```

These specific methods may be necessary during the construction of a distributed vector. These methods are never called by data assimilation methods which delegate the distributed variable initializations to models and observation managers.

#### Distributed PETSc vector example program

```
Vec x, y;
```
```c
int N = 10;

 ierr = VecCreateMPI(PETSC_COMM_WORLD, N, &x); CHKERRQ(ierr);
 ierr = VecCreateMPI(PETSC_COMM_WORLD, N, &y); CHKERRQ(ierr);
 ierr = VecSet(x, 3.0); CHKERRQ(ierr);
 ierr = VecSet(y, 1.0); CHKERRQ(ierr);

 ierr = VecAssemblyBegin(x); CHKERRQ(ierr);
 ierr = VecAssemblyEnd(x); CHKERRQ(ierr);
 ierr = VecAssemblyBegin(y); CHKERRQ(ierr);
 ierr = VecAssemblyEnd(y); CHKERRQ(ierr);

 ierr = VecAXPY(y, -1.0, x); CHKERRQ(ierr);
 ierr = VecDestroy(&x); CHKERRQ(ierr);
 ierr = VecDestroy(&y); CHKERRQ(ierr);
```

The same example using the `Vector< double, PETScPar >` class

```cpp
Vector< double, PETScPar > x, y;

x.Reallocate(10);

y.Reallocate(10);

x.Fill(3.0);

y.Fill(1.0);

Add(-1.0, x, y);
```

22.2.2.0.10 Distributed Dense Matrix

The class Seldon::Matrix<T, Prop, PETScMPIDense, Allocator> encapsulate a dense distributed PETSc matrix of type `MATMPIDENSE`. This class implement the same interface as a classic Seldon matrix.

The source files of the class Seldon::Matrix<T, Prop, PETScMPIDense, Allocator> are located in the seldon/matrix/ directory. Several methods are specific of Seldon::Matrix<T, Prop, PETScMPIDense, Allocator> class:

```cpp
template <class T, class Prop, class Allocator>

class Matrix<T, Prop, PETScMPIDense, Allocator>
{

    // Returns a reference on the inner petsc matrix.
    Mat& GetPetscMatrix();

    // Returns a const reference on the inner petsc matrix.
    const Mat& GetPetscMatrix() const;

    // Sets the MPI communicator.
    void SetCommunicator(MPI_Comm mpi_communicator);

    // Returns the MPI communicator of the current PETSc matrix.
    MPI_Comm Get Communicator() const;

    // Inserts or adds values into certain locations of a matrix.
    // Warning: These values may be cached, so 'Flush' must be called after all
    // calls to 'SetBuffer()' have been completed.
    void SetBuffer(int i, int j, T, InsertMode);

    // Assembles the PETSc matrix.
    void Flush() const;

    // Returns the range of row indices owned by this processor.
    // The matrix is laid out with the first \$n_1\$ rows on the first
    // processor, next \$n_2\$ on the second, etc. If the current
    // processor is \$i\$, this method returns \$n_k\$ in \$a_i\$ and
    // \$n_{k+1}\$ in \$a_j\$. If \$a_i\$ is set to PETSC_NULL on entry, it is not
    // modified by this function. Same is true for \$a_j\$.
    void GetProcessorRowRange(int& i, int& j) const;

    ...
};
```
These specific methods may be necessary during the construction of a distributed matrix. These methods are never called by data assimilation methods which delegate the distributed variable initializations to models and observation managers.

22.2.2.0.11 Distributed Sparse Matrix

The class Seldon::Matrix<T, Prop, PETScMPIAIJ, Allocator> encapsulate a sparse distributed PETSc matrix of type MATMPIAIJ. This class has the same interface as the Seldon::Matrix<T, Prop, PETScMPIIDense, Allocator> class introduced previously.

The source files of the class Seldon::Matrix<T, Prop, PETScMPIAIJ, Allocator> are located in the seldon/matrix/ directory

22.2.3 'PETScClampedBar' Distributed Model

Verdandi provides an implementation of the 'ClampedBar' model based on the distributed structures available in Seldon. The source files of the distributed 'PetscClampedBar' model are located in the verdandi/model/ directory.

22.2.3.0.12 The 'PETScClampedBar' Model

The clamped bar model describes the vibration of a bar clamped at one end. The bar is discretized with $N_x$ finite elements of the same length. With the hypothesis of "small displacements", it follows the linear system:

$$M \ddot{Y} + C \dot{Y} + KY = F_{\theta_f}$$

where $M$ is the mass matrix, $K$ is the stiffness matrix, $C$ is the damp matrix and $F(\theta_f) = \sin\left(\frac{\pi}{T_f}\right)M \theta_f(1...1)^T$ is the effort vector.

The clamped bar model is solved numerically using a Newmark scheme (middle point) for integration in time:

$$\dot{Y}_h^{n+1} = \frac{Y_{h+1}^{n+1} - Y_h^n}{\Delta t}$$

Algorithmically, it follows:

$$\dot{Y}_h^{n+1} = \frac{1}{2\Delta t}(Y_{h+1}^{n+1} - Y_h^n)$$

$$\text{Newmark}_1 = \frac{1}{2}K + \frac{1}{2\Delta t}C + \frac{1}{2\Delta t}M$$

$$\text{Newmark}_0 = -\frac{1}{2}K + \frac{1}{2\Delta t}C + \frac{2}{2\Delta t}M$$

$$\text{Newmark}_1 Y_h^{n+1} = \text{Newmark}_0 Y_h^n + \frac{2}{\Delta t}M \dot{Y}_h^n + F_{h+\frac{1}{2}}(\theta_f)$$

The matrices $M$, $\text{Newmark}_0$ and $\text{Newmark}_1$ are sparse distributed matrices of type Matrix<T, Prop, PETScMPIAIJ, Allocator>. The effort vector $F$ is a distributed vector of type Vector<double, PETScPar>:

```cpp
template <class T>
class PetscClampedBar : public VerdandiBase {

public:

... .

/// Mass matrix.
Matrix<T, General, PETScMPIAIJ> mass_; 
/// Newmark matrix 0.
Matrix<T, General, PETScMPIAIJ> newmark_0_; 
/// Newmark matrix 1.
Matrix<T, General, PETScMPIAIJ> newmark_1_; 
/// Force.
Vector<T, PETScPar> rhs_; 
```
Table 22.1: Distribution of the ‘PetscClampedBar’ state vector over processes.

<table>
<thead>
<tr>
<th>Processors</th>
<th>0</th>
<th>1</th>
<th>...</th>
<th>N process − 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathbf{Y} )</td>
<td>( \mathbf{Y}^0 = (Y_0, Y_{N_{\text{process}}}) )</td>
<td>( \mathbf{Y}^1 = (Y_{N_{\text{process}}}, Y_2, Y_{N_{\text{process}}}) )</td>
<td>...</td>
<td>( \mathbf{Y}^{N_{\text{process}} - 1} = (Y_{N_{\text{process}}}, Y_{N_{\text{process}}}, Y_{N_{\text{process}} - 1}) )</td>
</tr>
<tr>
<td>( \mathbf{\dot{Y}} )</td>
<td>( \mathbf{\dot{Y}}^0 = (\mathbf{\dot{Y}}<em>0, \mathbf{\dot{Y}}</em>{N_{\text{process}}}) )</td>
<td>( \mathbf{\dot{Y}}^1 = (\mathbf{\dot{Y}}<em>{N</em>{\text{process}}}, \mathbf{\dot{Y}}<em>2, \mathbf{\dot{Y}}</em>{N_{\text{process}}}) )</td>
<td>...</td>
<td>( \mathbf{\dot{Y}}^{N_{\text{process}} - 1} = (\mathbf{\dot{Y}}<em>{N</em>{\text{process}}}, \mathbf{\dot{Y}}<em>{N</em>{\text{process}}}, \mathbf{\dot{Y}}<em>{N</em>{\text{process}} - 1}) )</td>
</tr>
<tr>
<td>( \theta_f )</td>
<td></td>
<td></td>
<td></td>
<td>( \theta_f )</td>
</tr>
</tbody>
</table>

22.2.3.0.13 Management of the Model State Vector

The model state vector contains the displacement vector \( \mathbf{Y} \), the velocity vector \( \mathbf{\dot{Y}} \) and the parameter vector \( \theta_f \). The Table 22.1 gives the distribution of the state vector over processes.

The displacement vector \( \mathbf{Y} \) and the velocity vector \( \mathbf{\dot{Y}} \) are distributed vectors. The parameter vector \( \theta_f \) is a sequential vector stored on the process of rank \( N_{\text{process}} - 1 \). When \( \theta_f \) is updated, this process is responsible for sending the updated vector \( \theta_f \) to all other.

```cpp
template <class T>
class PetscClampedBar : public VerdandiBase
{
public:
    // Type of the model state vector.
    typedef Vector<T, PETScPar> state;
    // Type of the model parameter vector.
    typedef Vector<T> parameter;

    // Displacement.
    state displacement_0_;
    // Velocity.
    state velocity_0_;  
    // Force parameter.
    parameter theta_force_;  
    // Local size of state vector.
    int Nstate_local_;  
    // Model state.
    state state_;  

    GetState()
}
```

The model state vector is passed to the data assimilation method by local copy:

\[
X^i = (Y^i, \mathbf{\dot{Y}}^i), 0 \leq i \leq N_{\text{process}} - 2
\]

\[
X^{N_{\text{process}} - 1} = (Y^{N_{\text{process}} - 1}, \mathbf{\dot{Y}}^{N_{\text{process}} - 1}, \theta_f)
\]
22.2 Sequential Method applied to Parallel Model

When the model state vector is updated, it is necessary to update the vectors $\mathbf{Y}$ and $\dot{\mathbf{Y}}$. The process $N_{\text{process}} - 1$ must broadcast the updated vector $\theta_f$ to all other processes.

```cpp
template <class T>
void PetscClampedBar<T>::StateUpdated()
{
  int disp_start, disp_end;
  displacement_0_.GetProcessorRange(disp_start, disp_end);
  int state_start, state_end;
  state_.GetProcessorRange(state_start, state_end);
  for (int i = disp_start; i < disp_end; i++)
  {
    state_.SetBuffer(state_start++, displacement_0_(i));
    state_.SetBuffer(state_start++, velocity_0_(i));
  }
  if (rank_ == Nprocess_ - 1)
  {
    for (int j = 0; j < parameter_.GetSize(); j++)
      state_.SetBuffer(state_start++, theta_force_(j));
    state_.Flush();
    return state_;
  }
}
```

22.2.3.0.14 Covariance Matrix

The 'PetscClampedBarModel' provides a decomposition of the state error covariance matrix ($P$) as a product $LUL^T$. The matrix $U$ is a matrix of small size, it is implemented as a sequential dense matrix.

The matrix $L$ is a matrix of $M_{\text{State}} \times N_{\text{reduced}}$, it is implemented as a distributed dense matrix. Each row of $L$ is distributed over processes with the same distribution as the one of the model state vector.

22.2.3.0.15 Performance

Figure 22.7 introduces the performance of the 'PetscClampedBar' model.

22.2.4 'ReducedOrderUnscentedKalmanFilter'

22.2.4.0.16 Algorithm

**Sampling:**

$$x_h^{(i)} = x_h^a + L_h \sqrt{U_h^{-1} f(i)}, \quad 1 \leq i \leq p + 1$$

**Prediction:**

$$x_{h+1}^f = E_x(CM_h(x_h^{(x)}))$$

$$x_{h+1}^{(i)} = CM_h(x_h^{(i)})$$
Figure 22.7: Simulation time of the 'PetscClampedBar' model with $N_{\text{state}} = 5 \times 10^6$

$$L_{h+1} = [x_h^{(i)f}]D_a[V^*]^T \in \mathcal{M}_{n,p}$$

Update:

$$y_{h+1}^{(i)} = \mathcal{H}_{h+1}(x_{h+1}^{(i)f})$$

$$\{HL\}_{h+1} = [y_{h+1}^{(i)}]D_a[V^*]^T$$

$$U_{h+1} = P^T_{\alpha} + \{HL\}_{h+1}R_{h+1}^{-1}\{HL\}_{h+1} \in \mathcal{M}_p$$

$$x_{h+1}^{(s)} = \sum L_{h+1}[HL]_{h+1}^{-1}R_{h+1}^{-1}(y_{h+1} - E(y_{h+1}^{(i)}))$$

22.2.4.0.17 Distributed Structure in 'ReducedOrderUnscentedKalmanFilter'

The goal is that no variable of the model state size is allocated by any process. The concerned variables are $x^a$, $L \in \mathcal{M}_{N_{\text{state}}N_{\text{sigma}} \_\text{point}}$, and $[x^{(s)f}] \in \mathcal{M}_{N_{\text{state}},N_{\text{reduced}}}$:

- The state vector is a distributed dense vector whose management is delegated to the model (see Section 22.2.3.0.13). The model state access are performed thanks to Model::GetState and Model::StateUpdated methods.

- The sensitivity matrix $L$ is a row distributed dense matrix whose management is delegated to the model. The model is responsible for defining a row distribution compatible with the one of the state vector (see Section 22.2.3.0.14). The $L$ access is performed using Model::GetStateErrorVarianceProjector method.

- The matrix $[x^{(s)f}]$ must be defined as a distributed dense matrix which distribution is compatible with the one of the matrix $L$. Since the matrix $[x^{(s)f}]$ is peculiar to the ROUKF algorithm, its management can’t be delegated to the model. Thus, it is necessary to allocate this matrix in the data assimilation method. $[x^{(s)f}]$ construction requires two information: the type of the matrix (dense distributed) and the distribution over the processes ($[x^{(s)f}]$ distribution compatible with $L$ distribution).

$[x^{(s)f}]$ construction must be modular: ROUKF implementation must be the same in sequential and in parallel.

Specification of $[x^{(s)f}]$ type
In PETSc, the types of the structures are managed dynamically. Every matrix has the same static type 'Mat', the real type of the matrix is defined during the execution by the following function call:

```c
Mat A;
MatSetType(A, MATMPIDENSE);
```

Indeed, PETSc is written in C language, thus template function can’t be defined for linear algebra operations. In Seldon the type of the matrices and vectors are statics. The type of \( \mathbf{x}^{(*)} \) is the same as the one of \( \mathbf{L} \). It is no longer required to call any function to specify the type. (The management of the types in Verdandi is detailed in Section 22.2.1)

**Distribution of \( [\mathbf{x}^{(*)}] \)**

The distribution of \( [\mathbf{x}^{(*)}] \) is performed during the allocation. The distribution of \( \mathbf{L} \) must be provided to the constructor of matrix \( [\mathbf{x}^{(*)}] \).

**Allocation of a sequential matrix of \( \mathbb{M}_{m,n} \)**:

```c
Matrix<double> A;
A. Reallocate(m, n);
```

**Allocation of a parallel matrix of \( \mathbb{M}_{m,n} \) with a distribution of \( m_{\text{local}} \) rows on the local process**:

```c
Matrix<double> A;
A. Reallocate(m, n, mlocal);
```

In order to have the same code in sequential and in parallel, the following template function is defined:

```c
template <class Model, class T, class Prop, class Storage, class Allocator>
void Allocate(const Model& model, Matrix<T, Prop, Storage, Allocator>& A, int n, int m)
{
    A.Reallocate(m, n);
}
```

This template function is overloaded for distributed dense matrices:

```c
template <class Model, class T, class Prop, class Storage, class Allocator>
void Allocate(const Model& model, Matrix<T, Prop, PETScMPIDense, Allocator>& A, int n, int m)
{
    A.Reallocate(m, n, model.GetLocalM());
}
```

The allocation of \( [\mathbf{x}^{(*)}] \) is performed in ROUKF by the following call:

```c
Allocate(model_, X_i, Nstate, Nsigma_point);
```

So, the implementation of ROUKF is the same in sequential and in parallel. In sequential, this implementation does not required the addition of any methods to the model interface. In parallel, the model should defined the method `Model::GetLocalM()` which provides the number of local rows in the \( \mathbf{L} \) distribution.

### 22.2.4.0.18 Performance

The proposed implementation enables to apply the ROUKF algorithm to a distributed model. No variable of the model state size is allocated by any process. Thus the memory complexity is divided by the number of processes used.

Concerning the time performance, during the prediction step:

- the computation \( x_{h+1}^{(i)f} = \mathbb{M}_{h} (x_{h}^{(i)a}) \) has the same speed up as the one of the model.
- the computation \( \mathbf{L}_{h+1} = [\mathbf{x}_{h+1}^{(*)} | \mathbf{D}_a [\mathbf{V}^*]^T \) has a speed up equal to the number of processes.

During the update step:

- the computation \( \mathbf{x}_{h+1}^{(f)} = x_{h+1}^{f} + \mathbf{L}_{h+1} U_{h+1}^{-1} \{HL\}_h^{-1} R_{h+1}^{-1} (y_{h+1} - E_a (y_{h+1}^{(s)})) \) has a speed up equal to the number of processes.

The performances of the ROUKF algorithm are introduced in Figure 22.8.
Figure 22.8: Simulation time of the sequential ROUKF applied to the ‘PetscClampedBar’ model with $N_{state} = 5 \times 10^6$
CB (parallel) parameter estimation using ROUKF (sequential)
with $N_{state} = 10^6$, $N_{obs} = 10^3$ and $N_{sp} = 3$.

22.2.5 Example Programs

The example programs are located in the verdandi/example/petsc_clamped_bar directory.

22.2.5.0.19 Compilation

Dependencies

OpenMPI

- Download the following archive:
- Extract the archive and execute the following command in the source directory:

```
$ ./configure \
   CC=/usr/bin/gcc-4.2 \ 
   CXX=/usr/bin/g++-4.2 \ 
   F77=/usr/bin/gfortran \ 
   FC=/usr/bin/gfortran \ 
   CFLAGS=-m64 \ 
   CXXFLAGS=-m64 \ 
   FFLAGS=-m64 \ 
   FCFLAGS=-m64 \ 
   LDFLAGS=-m64 \
```

PETSc-3.3

- Download the following archive:
- Extract the archive and execute the following command in the source directory:
Example Programs

Compile the program `generate_observation.cpp`:

```bash
$ scons generate_observation mpi=yes
```

Then compile the program `reduced_order_unscented_kalman_filter.cpp`:

```bash
$ scons reduced_order_unscented_kalman_filter mpi=yes
```

22.2.5.0.20 Observation

Since no observations are given yet, we have to generate some. Execute the following command:

```bash
$ mpirun -n 2 generate_observation configuration/truth.lua
```

to run the model with the initial conditions described in truth.lua, without data assimilation. This should generate a result file `truth-state_forecast.bin` in the directory `example/result`. This file store the state (displacement, velocity, \( \theta_f \)) trajectory. The generated state (displacement, velocity, \( \theta_f \)) will serve as observations for the assimilation.

22.2.5.0.21 Data Assimilation with 'ReducedOrderUnscentedKalmanFilter'

To use the ROUKF method, execute the following command:

```bash
$ mpirun -n 4 reduced_order_unscented_kalman_filter configuration/assimilation.lua
```

All processes are assigned to the model. The results should be the same as those obtained in sequential.

22.3 Parallel Method applied to Parallel Model

22.3.1 Parallel 'ReducedOrderUnscentedKalmanFilter'

22.3.1.0.22 MPI Communicator

The objective is to apply the parallel ROUKF algorithm introduced in Section 22.1 on a distributed model. We want to implement the capabilities in Verdandi to instantiate several models in parallel, and to assign several processes to each model task. A possible solution consist in using the grid topology provided by MPI:

- each process is defined in a MPI processor grid.
- each process is identified by its coordinates in the grid. Figure 22.9 represents an example of a mapping table of process rank and their corresponding grid coordinates.
- for each row of the grid, a communicator containing the subgrid that includes the processes of the row is created. For instance, in Figure 22.9, the group of the second row communicator is composed of processes
4, 5, 6 and 7 from \texttt{MPI\_COMM\_WORLD}. Process 0 in second row communicator is the same as process 4 in \texttt{MPI\_COMM\_WORLD}, process 1 the same as process 5...

- for each column of the grid, a communicator containing the subgrid that includes the processes of the column is created.

Each column communicator correspond to an instance of model. In Figure 22.9, there is 4 instance of model. The first column communicator, composed of processes 0, 4, 8 and 12, is assigned to the first model instance. Then, the second column communicator is assigned to the second model instance...

Some changes about the configuration of MPI communicators in the distributed model may be required. Indeed, the distributed model are not allowed to performed computations in the global MPI communicator \texttt{MPI\_COMM\_WORLD}. To enable several parallel model instances, \texttt{MPI\_COMM\_WORLD} has to be replaced by a communicator of disjoint process sets in which each of the model instance operates. The column communicators to be assigned to the model instances are generated by the data assimilation method. These communicators are assigned to the model instances thanks to the following method:

```cpp
namespace Verdandi {

  // This class is a model template.
  class ModelTemplate : public VerdandiBase {
    
    public:
      ...

    // Parallel model.
    #ifdef VERDANDI_WITH_MPI
    void SetMPICommunicator(MPI_Comm& mpi_communicator);
    #endif
    ...
  }
}
```

The row communicators enable the parallel ROUKF algorithm to update its distributed variables $x^t$, $L \in \mathcal{M}_{n,p}$ and $[x^{t+1}]_f \in \mathcal{M}_{n,r}$.

22.3.1.0.23 Algorithm

This Section describes the parallel ROUKF algorithm applied to a parallel model. The processes are mapped to a process grid by using row-major order (see Section 22.3.1.0.22).

Initialization

- First column of the MPI grid (processes 0, 4, 8, 12; instance of model 0) allocates matrix $L$. 

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>(0,0)</td>
<td>(0,1)</td>
<td>(0,2)</td>
<td>(0,3)</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>(1,0)</td>
<td>(1,1)</td>
<td>(1,2)</td>
<td>(1,3)</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>(2,0)</td>
<td>(2,1)</td>
<td>(2,2)</td>
<td>(2,3)</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>(3,0)</td>
<td>(3,1)</td>
<td>(3,2)</td>
<td>(3,3)</td>
</tr>
</tbody>
</table>
22.3 Parallel Method applied to Parallel Model

Sampling

- First column of the MPI grid computes \( x_h^{(i)a} = x_h^{a} + L_h \sqrt{U_h^{-1}} f(i) \), \( 1 \leq i \leq p + 1 \).
- First column of the MPI grid distributes particles \( x_h^{(i)a} \) over all columns.

Prediction

- Each column of the MPI grid computes in parallel \( x_h^{(i)f} = \mathcal{H}_h(x_h^{(i)a}) \) with its local particles.
- Each column of the MPI grid sends its local particles \( x_h^{(i)f} \) to the first column.
- Each column of the MPI grid computes in parallel \( y_h^{(i)} = \mathcal{H}_{h+1}(x_h^{(i)f}) \).
- Each column of the MPI grid sends its local particles \( y_h^{(i)} = \mathcal{H}_{h+1}(x_h^{(i)f}) \) to the first column.
- First column of the MPI grid computes \( L_{h+1} = [x_h^{(i)f} D_a[V^*]^T] \).

Update

- First column of the MPI grid computes \( \{HL\}_{h+1} = [y_{h+1}^T] D_a[V^*]^T \).
- First column of the MPI grid computes \( U_{h+1} = P_a^W + \{HL\}_{h+1} R_{h+1}^{-1} \{HL\}_{h+1} \).
- First column of the MPI grid computes \( x_h^{a} = x_h^{f} + L_{h+1} U_{h+1}^{-1} \{HL\}_{h+1} R_{h+1}^{-1} (y_{h+1} - E_a(y_{h+1})) \).

22.3.2 Example Programs

The example programs are located in the verdandi/example/petsc_clamped_bar directory.

22.3.2.0.24 Compilation

First of all, the preprocessor variable \texttt{VERDANDI\_WITH\_MPI} has to be defined in files \texttt{reduced\_order\_extended\_kalman\_filter.cpp} and \texttt{reduced\_order\_unscented\_kalman\_filter.cpp}:

```cpp
#include "Verdandi.hxx"
#include "seldon/SeldonSolver.hxx"
#include "model/PetscClampedBar.hxx"
#include "observation_manager/PetscLinearObservationManager.hxx"
#include "method/ReducedOrderUnscentedKalmanFilter.hxx"
```
int main(int argc, char ** argv) {
    VERDANDI_TRY;
    ...
}

Compile the program **generate_observation.cpp**:

```bash
$ scons generate_observation mpi=yes
```

Then compile the program **reduced_order_unscented_kalman_filter.cpp**:

```bash
$ scons reduced_order_unscented_kalman_filter mpi=yes
```

### 22.3.2.0.25 Observation

Since no observations are given yet, we have to generate some. Execute the following command:

```bash
$ mpirun -n 2 generate_observation configuration/truth.lua
```

to run the model with the initial conditions described in truth.lua, without data assimilation. This should generate a result file **truth-state_forecast.bin** in the directory **example/result**. This file store the state (displacement, velocity, \( \theta_f \)) trajectory. The generated state (displacement, velocity, \( \theta_f \)) will serve as observations for the assimilation.

### 22.3.2.0.26 Data Assimilation with 'ReducedOrderUnscentedKalmanFilter'

The parameters of the ROUKF method are described in the configuration file **configuration/assimilation.lua**.

It is necessary to define the dimension of the MPI grid. The number of model instances and the number of processes assigned to each model instance must be defined:

**assimilation.lua**

```lua
-- Simulation with assimilation using ROUKF.
reduced_order_unscented_kalman_filter = {
    ...
    mpi_grid = {
        -- The number of processes for each model task.
        Nrow = 2,
        -- The number of model tasks.
        Ncol = 3
    }
}
```

To use the ROUKF method, execute the following command:

```bash
$ mpirun -n 6 reduced_order_unscented_kalman_filter configuration/assimilation.lua
```

**Warning:** The number of processes must be equal to \( mpi_grid.Nrow \times mpi_grid.Ncol \).

The results should be the same as those obtained in sequential.

### 22.4 Performance

We can see in Figure 22.10 that when the number of processes is less than or equal to four, it is more in efficient to instantiate only one model and to assign all processes to this instance. On the other hand, when the number of processes increases, the most effective is to own several model instances:
Figure 22.10: PetscClampedBar parameter estimation using ROUKF (parallel) with $N_{state} = 10^5$ and $N_{observation} = 10^4$
– when the number of processes is equal to sox, the most efficient is to define two model instances running on three processes.

– when the number of processes is equal to sox, the optimal configuration is to have two model instances running on four processes.

As intended, this second level of parallelism provides a better scalability. At the first level of parallelism, when the efficiency of ROUKF algorithm decreases, this second level of parallelism contributes to better take advantage of the available computation resources.
Part V

Example Models
Chapter 23

Example Models

If you want to write an interface to your model, please refer to the following pages:

- Plugging in Verdandi: Model in C++;
- Plugging in Verdandi: Model in Python.

In addition, several example models have been implemented in Verdandi. Follow the links below for a description of the available models:

- Quadratic model, which is also implemented in Python;
- Shallow-water model;
- Clamped bar;
- Lorenz model.

For details about how to apply data assimilation methods on these models see Example Programs.
Chapter 24

Clamped Bar

Below is the physical description of the clamped bar model. For details about how to apply data assimilation methods on this model see Clamped Bar.

24.1 The Clamped Bar model

The clamped bar model describes the vibration of a bar clamped at one end. The bar is discretized with $N_x$ finite elements of the same length. With the hypothesis of "small displacements", it follows the linear system:

$$ M\ddot{\theta} + C\dot{\theta} + K\theta = F $$

where $M$ is the mass matrix, $K$ is the stiffness matrix, $C = \alpha M + \beta M$ is the damp matrix and $F$ is the effort vector.

The clamped bar model is solved numerically using a Newmark scheme (middle point) for integration in time:

$$
\begin{align*}
\ddot{Y}_{h+\frac{1}{2}} &= \frac{\dot{Y}_{h+1} + \dot{Y}_h}{2} = \frac{Y_{h+1} - Y_h}{\Delta t} \\
Y_{h+\frac{1}{2}} &= \frac{Y_{h+1} + Y_h}{2} = \frac{Y_{h+1} - Y_h}{\Delta t}
\end{align*}
$$

Algorithmically, it follows:

$$
\begin{align*}
Y_{h+1} &= \left(\frac{3}{2} \Delta t\right)Y_h + \frac{\Delta t^2}{2} (K\theta + C\theta + 2M\dot{\theta}_h)Y_{h+1} = -\frac{\Delta t}{2} K\theta + \frac{1}{4} C\theta + \frac{2}{\Delta t^2} M\dot{\theta}_h + \frac{3}{2} M\theta_h + \frac{\Delta t^2}{2} (\theta_f)
\end{align*}
$$

24.2 Simulation

During the simulation, a force $F(\theta_f) = \sin\left(\frac{\pi}{T}\right)M\theta_f (1...1)^T$ is applied to the bar at each point.

For each parameter ($\theta_m$, $\theta_c$, $\theta_k$, $\theta_f$), the bar can be divided into several regions of the same length.

For example, if we supposed that the bar is divided, for the parameter $\theta_f$, into four regions of the same length, with $\theta_f = (0.8, 1.5, 0.6, 2.0)$, the output should look something like this:

One can define several parameters:

1. bar_length: the length of the bar;
2. Nx: the number of elements;
3. Delta_t: the time step;
4. final_time: the duration of the simulation;
5. Young_modulus: the Young’s modulus;
6. mass_density: the mass density;
7. theta_mass : vector that contains the initial values of $\theta_m$;
8. theta_damp : vector that contains the initial values of $\theta_c$;
9. alpha : damp coefficient $\alpha$;
10. beta : damp coefficient $\beta$;
11. theta_stiffness : vector that contains the initial values of $\theta_k$;
12. theta_force : vector that contains the initial values of $\theta_f$.

The clamped-bar model is implemented in ClampedBar.hxx and ClampedBar.cxx. The class ClampedBar is a template class: ClampedBar<T>. T is the type of the elements to be stored (e.g. double).

The state may contain:

- "displacement": the vertical displacement along y;
- "velocity": the vertical velocity along y;
- "theta_mass": the vector $\theta_m$;
- "theta_damp": the vector $\theta_c$;
- "theta_stiffness": the vector $\theta_k$;
- "theta_force": the vector $\theta_f$.

The displacement and the velocity at the first point are equal to zero since the bar is clamped. These zeroes are removed from the state. The size of the state is therefore $2 (Nx - 1) + Ntheta_mass + Ntheta_damp + Ntheta_stiffness + Ntheta_force$.
Chapter 25

Lorenz model

The Lorenz model is a 3-dimensional chaotic system. The equations of the model are:

\[
\begin{align*}
\partial_t x &= Pr(y - x) \\
\partial_t y &= x(Ra - z) - y \\
\partial_t z &= xy - bz
\end{align*}
\]

where \( Pr \) is the Prandtl number, \( Ra \) the Rayleigh number, and \( b \) is positive number.

The Lorenz model is implemented in Lorenz.hxx and Lorenz.cxx. The class Lorenz is a template class: Lorenz<T>. \( T \) is the numerical type of the variables (e.g., double).

The state vector is \((x, y, z)\).
Chapter 26

Quadratic Model

In order to define the quadratic model, we introduce:

- $n$ matrices $S_i \in \mathbb{R}^{n \times n}$ for the quadratic part of the model,
- one matrix $L \in \mathbb{R}^{n \times n}$ for the linear part of the model, and
- a vector $b \in \mathbb{R}^n$ for the constant part of the model.

The equation of the $i$th state component of the quadratic model ($i \in \{1, \ldots, n\}$) is

$$\frac{dx_i}{dt} = x^T S_i x + L_i x + b_i,$$

where $L_i$ is the $i$th line of $L$.

In case the model has no quadratic part, the state equation is simply

$$\frac{dx}{dt} = L x + b.$$
The shallow-water equations (also called Saint Venant equations) are a set of hyperbolic partial differential equations that describe the flow below a pressure surface in a fluid. It is suitable for flows with a free surface and small depth. For instance, these equations can be applied to model the behavior of a lake or a river. The model describes the evolution of the water height $h(x, y)$ and the horizontal velocity $(u(x, y), v(x, y))$. A simplified expression of the model is:

$$
\partial_t h + \partial_x (hu) + \partial_y (hv) = 0
$$

$$
\partial_t (hu) + \partial_x (huu) + \partial_y (huv) + \frac{1}{2} g \partial_x h^2 = 0
$$

$$
\partial_t (hv) + \partial_x (huv) + \partial_y (hvv) + \frac{1}{2} g \partial_y h^2 = 0
$$

One can define several boundary conditions:

1. an incoming flow rate $q_b = h_b u_b$ (or $q_b = h_b v_b$ along $y$), with the sign of $u_b$ (or $v_b$) being determined by the considered boundary (for instance, $u \leq 0$ on the right boundary);
2. a homogeneous Neumann condition for $(h, u, v)$;
3. an impermeability condition: the flow rate is zero at the boundary;
4. a fixed height $h_b$.

The shallow-water model is implemented in ShallowWater.hxx and ShallowWater.cxx. The class ShallowWater is a template class: ShallowWater<T>. T is the numerical type of the variables (e.g., double).

The state contains:

- the water height (stored in the matrix $h$);
- the horizontal velocity along x (stored in matrix $u$);
- the horizontal velocity along y (stored in matrix $v$).

For details about how to apply data assimilation methods on this model see Shallow Water.
Part VI

Example Observation Managers
Chapter 28

Observations

In order to plug your own observation manager (i.e., the class that provides the observations), you should refer to the page Plugging in Verdandi: Observations, or directly see the required interface shown by the Observation-ManagerTemplate (in C++).

You may also use or get help from the two observation managers implemented in Verdandi:

- linear observation manager, which is also available in Python;
- grid to network observation manager.

An observation aggregator which allows to compute aggregated (in time) and flattened observations is also available:

- observation aggregator.

The observation aggregator is coupled with the linear observation manager. One can use the observation aggregator in his own observation manager. For more detail about how to write a new observation manager see the page Plugging in Verdandi: Observations.
Chapter 29

Grid To Network Observation Manager

This observation manager defines a grid-to-network observation operator $\mathcal{K}_h = H$, where $H$ is a time-independent matrix.

It is implemented in GridToNetworkObservationManager.hxx and GridToNetworkObservationManager.cxx. The class GridToNetworkObservationManager is a template class: GridToNetworkObservationManager<T>. $T$ is the numerical type (e.g., double).
Chapter 30  

Linear Observation Manager

This observation manager defines a linear observation operator \( \mathcal{H}_h = H \), where \( H \) is a time-independent matrix. The class `LinearObservationManager` is implemented in `LinearObservationManager.hxx` and `LinearObservationManager.cxx`. It is a template class: `LinearObservationManager<T>`. \( T \) is the numerical type (e.g., `double`).

### 30.1 Management of the observations

#### 30.1.1 How the observations are stored

The observations should be available in binary format. One should define the path to the file storing the observations in the configuration file:

```plaintext
observation = {
    file = observation_file,
    ...
}
```

One should also define the period with which observations are available.

At each time \( t_h \), the provided observations can be an approximation of either the whole model state \( x_t \) or the observations \( \mathcal{H}_h(x_t) \) only.

#### 30.1.2 Observation contributions

When a data assimilation method requests for observations to the `LinearObservationManager` at a given time \( t_h \), it is possible that no observation is available at this time.

The `LinearObservationManager` associates to each available observation \( y_i \) a contribution \( \alpha_i \). The observation aggregator enables to compute, at each time \( t_h \), the contribution \( \alpha_i \) according to several rules: 'step', 'interpolation', 'triangle'.

### 30.2 Observation operator

There are two ways to define the observation operator. One can use a Lua table to define the observation operator:

```plaintext
observation = {
    ...
}
```
operator = { 
  ... 
  value = \{1, 0, 0, 0 \\
  0, 0, 1, 0\} 
}, 

It is also possible to define the observation operator in binary format. One should define the path to the file storing the observation operator.

### 30.3 Observation error variance

The observation error variance $R_h$ is supposed to be a scaled identity matrix.
Chapter 31

Observation Aggregator

The Observation Aggregator allows to compute aggregated and flattened observations. The class ObservationAggregator is implemented in ObservationAggregator.hxx and ObservationAggregator.cxx. It is a template class: ObservationAggregator<T>. T is the numerical type (e.g., double).

When a data assimilation method requests for observations to an observation manager at a given time $t_h$, it could be possible that no observation is available at this time.

An observation manager can associate to each available observation $y_i$ a contribution $\alpha_i$. The Observation Aggregator enables to compute, at each time $t_h$, the contribution of each observation according to several rules: 'step', 'interpolation', 'triangle'.

31.1 Step interpolation

In this mode, any observation is selected if its date belongs to a left-closed and right-open interval centered on $t_h$.

```plaintext
aggregator = {
  type = "step",
  width_left = 0.05,
  width_right = 0.07,
  ...
}
```

31.2 Interpolation

In this mode, the closest left observation from $t_h$ and the closest right observation are selected. The two observations are then interpolated. One should define an observation interval. It is assumed that the observations outside this interval have no contribution.

```plaintext
aggregator = {
  type = "interpolation",
  -- Observation interval.
  width_left_upper_bound = 1.,
  width_right_upper_bound = 1.,
  ...
}
```

31.3 Triangle interpolation

In this mode, one should define an observation interval (left-closed and right-open). All observations available in this interval are considered. The observations outside this interval have no contribution. They are interpolated by a
triangle centered on $t_h$. The widths of the triangles may not be constant.

```
observation = {
  ...
  width_file = "configuration/width.bin",
  aggregator = {
    type = "triangle",
    width_property = "per-observation",
    -- Observation interval.
    width_left_upper_bound = 1.,
    width_right_upper_bound = 1.,
    ...
  }
  ...
```
Part VII

Plugging in Verdandi
Chapter 32

Plugging in Verdandi

32.1 How To Plug a Model and Observations in Verdandi

This section of the documentation explains how to plug an external model and observations in Verdandi. The first step is to build an interface to the model. The second step is either having an existing observation manager read the observations, or writing a new observation manager.

32.2 Reference Notation

The tables below list the variables involved in data assimilation. For each variable, the name of the C++ type (a typedef) and the recommended C++ variable name are given.

For details about the mathematical description of these variables see section mathematical notation.

32.2.1 Model

<table>
<thead>
<tr>
<th>Variable(s)</th>
<th>Typedef</th>
<th>C++ Variable(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x, x^f, x^a$</td>
<td>state</td>
<td>$x, x_f, x_a$</td>
</tr>
<tr>
<td>$P, P^f, P^a$</td>
<td>state_error_variance</td>
<td>$P, P_f, P_a$</td>
</tr>
<tr>
<td>Row of $P, P^f, P^a$</td>
<td>state_error_variance_row</td>
<td>$P_row, P_f_row, P_a_row$</td>
</tr>
<tr>
<td>$Q$</td>
<td>error_variance</td>
<td>$Q$</td>
</tr>
<tr>
<td>$e^m$</td>
<td>systematic_error</td>
<td>$e^m$</td>
</tr>
<tr>
<td>$M$</td>
<td>tangent_linear_operator</td>
<td>$M$</td>
</tr>
</tbody>
</table>

matrix_state_observation determines the type of matrices of size $n \times m$. 
### 32.2.2 Observation Manager

<table>
<thead>
<tr>
<th>Variable(s)</th>
<th>Typedef</th>
<th>C++ Variable(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y$</td>
<td>observation</td>
<td>$y$</td>
</tr>
<tr>
<td>$R$</td>
<td>observation error variance</td>
<td>$R$</td>
</tr>
<tr>
<td>$\bar{e}_o$</td>
<td>observation systematic error</td>
<td>$\bar{e}_o$</td>
</tr>
<tr>
<td>$H$</td>
<td>tangent linear operator</td>
<td>$H$</td>
</tr>
</tbody>
</table>

### 32.2.3 Notes

In the assimilation algorithms, the same typedefs are defined with `model_` or `observation_` prepended. There are two exceptions: nothing is prepended to `matrix_state_observation` and `observation(y)`. The typedefs should be public in the model class and the observation class, so that the assimilation method may access to them.

In the assimilation methods, the following variables are also used.

<table>
<thead>
<tr>
<th>Variable(s)</th>
<th>Meaning</th>
<th>C++ Variable(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l$</td>
<td>Number of model parameters (size of $p$)</td>
<td>$N_{\text{parameter}}$</td>
</tr>
<tr>
<td>$m$</td>
<td>Number of observations (size of $y$)</td>
<td>$N_{\text{observation}}$</td>
</tr>
<tr>
<td>$n$</td>
<td>Size of the state vector $x$</td>
<td>$N_{\text{state}}$</td>
</tr>
</tbody>
</table>
Chapter 33

Plugging a Model

33.1 Create your own model

Start from the ModelTemplate. The complete interface of the model class is described here. With a complete interface, one can apply any data assimilation method in Verdandi. But for a given data assimilation method, not all methods are required. Use the Model requirements page, or the Python script bin/methods_requirement to know precisely which methods you will need to implement in your model.

- In the model directory, copy the ModelTemplate and rename it to "MyModel". E.g., under Linux or MacOS, in command line:
  $ cd verdandi/model/ $ cp ModelTemplate.hxx MyModel.hxx $ cp ModelTemplate.cxx MyModel.cxx
- Open the files MyModel.xxx, rename the preprocessing variables.

```c
#ifndef VERDANDI_FILE_MODEL_MODELTEMPLATE_*XX
#define VERDANDI_FILE_MODEL_MODELTEMPLATE_*XX

#include "Model.hxx"

class MyModel : public Model {

public:

  // ...

private:

  // ...

};

#endif
```

- Replace all "ModelTemplate" occurrences with "MyModel".
- In a first step, it is advised to write the interface for the ForwardDriver method which requires only a limited number of methods to be implemented.

Create a directory my_model in the example directory. Copy in my_model the following files:
- example/quadratic_model/SConstruct, example/quadratic_model/forward.cpp
- and the configuration file example/quadratic_model/configuration/truth.lua. E.g.:
  $ cd verdandi/example/ $ mkdir -p my_model/configuration $ cp quadratic_model/{SConstruct,forward.cpp} my_model/ $ cp quadratic_model/configuration/truth.lua my_model/configuration/
- In the file forward.cpp, replace all "ModelTemplate" occurrences with "MyModel".
- Compile the program forward.cpp.
  $ scons forward
- Run the forward program.
  $ ./forward configuration/truth.lua

At run time, if a method is needed in the model class, and if this method is not implemented, the program will launch an exception. For instance, if the method HasFinished is undefined, an exception will be thrown to report you this error. Then, see the contract of this method (HasFinished) and implement it. You can also have a look into the example models in verdandi/model/.
33.2 Plugging an existing model

Usually, one wants to create an interface for an existing model and to rely on existing model compilation tools.

We recommend the following procedure:

1. Create a class which encapsulates the existing model and which implements the Verdandi model interface (follow the same steps as in the previous section, Create your own model).

Example

```cpp
class ExampleModel
{
    private:
        int Ndisplacement;
        double* displacement;
        int Nvelocity;
        double* velocity;
        ...
    public:
        // Constructor and destructor.
        ExampleModel();
        ~ExampleModel();
        int GetNdisplacement();
        double* GetDisplament();
        int GetNvelocity();
        double* GetVelocity();
        ...
};

class MyModel: public Verdandi::VerdandiBase
{
    typedef Vector<double> state;
    private:
        ExampleModel model_; // Encapsulate the existing model
        state x_; // State
    public:
        // Constructor and destructor.
        MyModel();
        ~MyModel();
        ...
        // Access methods.
        int GetNstate();
        state& GetState();
        void StateUpdated();
        ...
};
```

For each method needed in the model class, see the contract provided by the ModelTemplate and implement it. For example, the GetNstate method contract is available here and its corresponding implementation could be:

```cpp
int MyModel::GetNstate()
{
    return model_.GetNdisplacement() + model_.GetNvelocity();
}
```

Accessing the Model State by Copy
Access to the model state vector can be performed by copy or reference through the methods `GetState` and `StateUpdated`.

Here is described an implementation of these methods which enables to access the model state by copy.

```cpp
MyModel::state& MyModel::GetState()
{
    x_.Reallocate(model_.GetNdisplacement() + model_.GetNvelocity());
    int position = 0;
    for (int i = 0; i < model_.GetNdisplacement(); i++)
        x_[position++] = model_.GetDisplacement()[i];
    for (int i = 0; i < model_.GetNvelocity(); i++)
        x_[position++] = model_.GetVelocity()[i];
    return x_;
}

void MyModel::StateUpdated()
{
    int position = 0;
    for (int i = 0; i < model_.GetNdisplacement(); i++)
        model_.GetDisplacement()[i] = x_[position++];
    for (int i = 0; i < model_.GetNvelocity(); i++)
        model_.GetVelocity()[i] = x_[position++];
}
```

### Accessing the Model State by Reference

In case of large-scale model, it is advised to pass the model state vector by reference. If the state vector is not contiguously stored in memory, the type of the model state vector has to be changed from a `Seldon` dense vector to a `Seldon` collection vector (a structure for distributed vectors):

```cpp
class MyModel: public Verdandi::VerdandiBase
{
    typedef Vector<double, Collection> state;
    ...
}
```

Here is described an implementation of the `Initialize` method to show how the `Seldon` collection vector can be initialized.

```cpp
void MyModel::Initialize(string configuration_file)
{
    ...
    Vector<double> displacement;
    // This method sets the inner vector displacement pointer
    // to the 'ModelExample' array containing elements.
    displacement.SetData(model_.GetDisplacement());
    Vector<double> velocity;
    // This method sets the inner vector velocity pointer
    // to the 'ModelExample' array containing elements.
    velocity.SetData(model_.GetVelocity());

    // Adds a vector to the list of vectors.
    x_.AddVector(displacement, "displacement");
    // The vector is "appended" to the existing data.
    x_.AddVector(velocity, "velocity");

    // Clears the vector without releasing memory.
    // On exit, the vector is empty and the memory has not been
    // released.
    displacement.Nullify();
    velocity.Nullify();

    ...
}
```

For the `GetState` method, copies are no longer required since the `ModelTemplate` state is a collection of vectors in which the inner pointer is set to the 'ModelExample' array containing state elements. The corresponding implementation should look something like this:
MyModel::state& MyModel::GetState()
{
    return x_;  
}

For the StateUpdated, calculations when the update of the model state is done are no longer required.

void MyModel::StateUpdated()
{
}

2. Create an object file from this class using your own compilation tools.

3. Add this object file to the Verdandi SConstruct dependency_list variable. Edit SConstruct file from the current project directory so that it looks like this:

```python
import os

# Put the path to Verdandi. # Also editable from command line with option "verdandi". verdandi_path = "MY_VERDANDI_PATH/verdandi-[version]"

dependency_list = ["MyModel.o"]
execfile(os.path.join(verdandi_path, "share/SConstruct"))
```

4. In the same way, update the Verdandi SConstruct include_path variable.

5. If your model has other dependencies, add the required include paths and library paths to the include_path and library_path variables.

6. Compile your program with SCons.
Chapter 34

Plugging a Python model

Start from the PythonModelTemplate. The complete interface of the model class is described here. With a complete interface, one can apply any data assimilation method in Verdandi. But for a given data assimilation method, not all methods are required. Use the Model requirements page, or the Python script bin/methods_requirements to know precisely which methods you will need to implement in your model.

- The Python model interface requires the use of Numpy, a package for scientific computing in Python. You will need to install it (and its headers).
  Under Windows, you may need to include manually the path to Numpy headers. You would then add the following line in the SConstruct file:

```python
include_path = "numpy_path"
```

where numpy_path should be replaced with the directory containing Numpy headers. This line must be inserted before the execfile statement.

- In the model directory, copy the PythonModelTemplate and rename it to "MyModel". E.g., under Linux or MacOS, in command line:
  
  ```bash
  $ cd verdandi/model/ $ cp PythonModelTemplate.py MyModel.py
  ```

- Rename the name of the class to MyModel.

- In a first step, it is advised to write the interface for the ForwardDriver method which requires only a limited number of methods to be implemented.
  
  Create a directory my_model in the example directory. Copy in my_model the following files: example/quadratic_model/SConstruct, example/quadratic_model/forward_python.cpp and the configuration file example/quadratic_model/configuration/truth.lua. E.g.,

  ```bash
  $ cd verdandi/example/ $ mkdir -p my_model/configuration $ cp quadratic_model/{SConstruct,forward_python.cpp} my_model/ $ cp quadratic_model/configuration/truth.lua my_model/configuration/
  ```

- In the file my_model/configuration/truth.lua, replace all the values in the python_model section with the corresponding ones on your model:

```lua
python_model = {
    module = "MyModel", -- name of the Python module containing the model class. directory = "model/", -- directory where the Python module is located. class_name = "MyModel" -- name of the Python model class.
}
```

- Compile the program forward_python.cpp.

  ```bash
  $ scons forward_python
  ```
– Run the program.

$ ./forward_python configuration/truth.lua

At run time, if a method is needed in the model class, and if this method is not implemented, the program will launch an exception. For instance, if the method `HasFinished` is undefined, an exception will be thrown to report you this error. Then, see the contract of this method (`HasFinished`) and implement it. You can also have a look into the example models in `verdandi/model/`.
Chapter 35

Methods requirements for models
Chapter 36

Methods requirements for observation managers
Chapter 37

Plugging the Observation Manager

If you have already plugged your own model in Verdandi, you can now work on the observations. Start from the ObservationManagerTemplate. The complete interface of the observation manager class is described here. With a complete interface, one can apply any data assimilation method in Verdandi. But for a given data assimilation method, the full interface may not be required. Use the Observation managers requirements page, or the Python script bin/methods_requirement to know precisely which interface you will need to implement for your observation manager.

- In the directory observation_manager, copy the ObservationManagerTemplate and rename it to “MyObservationManager”. E.g., under Linux or MacOS, in command line:
  
  
  $ cd verdandi/observation_manager/ $ cp ObservationManagerTemplate.hxx MyObservationManager.hxx $ cp ObservationManagerTemplate.cxx MyObservationManager.cxx

- Open the files MyObservationManager.xx, rename the preprocessing variables.

  #ifndef VERDANDI_FILE_MODEL_OBSERVATIONMANAGERTEMPLATE_*XX
  #define VERDANDI_FILE_MODEL_OBSERVATIONMANAGERTEMPLATE_*XX

  to

  #ifndef VERDANDI_FILE_MODEL_MYOBSERVATIONMANAGER_*XX
  #define VERDANDI_FILE_MODEL_MYOBSERVATIONMANAGER_*XX

- Replace all occurrences of “ObservationManagerTemplate” with “MyObservationManager”.

- You can begin with writing the interface for the method OptimalInterpolation which requires a basic interface. In some testing directory, copy the following files: example/quadratic_model/SConstruct, example/quadratic_model/optimal_interpolation.cpp and the configuration file example/quadratic_model/configuration/assimilation.lua. E.g., assuming you already created a directory verdandi/example/my_model/ as proposed in the page describing the plugging of a model:

  $ cd verdandi/example/ $ cp quadratic_model/{SConstruct,optimal_interpolation.cpp} my_model/ $ cp quadratic_model/configuration/assimilation.lua my_model/configuration/

- In the file optimal_interpolation.cpp, replace all "ObservationManagerTemplate" occurrences with "MyObservationManager". You will probably change the model as well.

- Compile the program optimal_interpolation.cpp.

  $ scons optimal_interpolation

- Run the optimal interpolation program.

  $ ./optimal_interpolation configuration/assimilation.lua

  At run time, if a method is needed in the model class, and if this method is not implemented, the program will launch an exception. For instance, if the method HasObservation is undefined, an exception will be thrown
to report you this error. Then, see the contract of this method and implement it. You can also have a look into the example managers in verdandi/observation_manager/. Be careful with the Linear-ObservationManager example, as it is way more advanced that what you need for a simple approach.
Part VIII

Tools
Chapter 38

Tools

Verdandi includes:

- a perturbation manager which allows the user to perturb a variable by using a random number generator. An example of its use in a method can be found in the Monte Carlo method.

- an optimization solver to carry out nonlinear optimization. An example of its use in a method can be found in the Four Dimensional Variational (4DVar) method.
Chapter 39

Perturbation manager

A perturbation manager in Verdandi is a tool to generate pseudo-random numbers. Two random numbers generator libraries are at this time interfaced: Newran and TRNG. One may find the implementations in the base class BasePerturbationManager.hxx and BasePerturbationManager.cxx and the derived classes for each library: NewranPerturbationManager.hxx and NewranPerturbationManager.cxx for Newran, TRNGPerturbationManager.hxx and TRNGPerturbationManager.cxx for TRNG.

39.1 Configuration options

In models, the perturbation manager can be used to perturb a variable with randomly generated numbers. Here are the main parameters.

1. distribution: the type of the distribution for the random vector;
2. mean: mean of the distribution
3. variance: covariance matrix of the distribution;
4. parameter: vector either empty or that contains two values for clipping the distribution;
5. correlation: vector that contains the correlation between sub-vectors;
6. option: defines when the perturbations are to be applied (at the first step only or at every step).

Here is one example derived from the Quadratic Model configuration file. We want to apply a perturbation on the constant term.

```plaintext
quadratic_model = {
    definition = {
        constant = {1., -1.}
    },

    uncertainty = {
        constant = {
            distribution = "Normal",
            mean = {0.5, 0.5},
            variance = {1., 0., 0., 1.},
            parameter = {-1., 1.},
            correlation = {}
        }
    }
}
```
Keep in mind that the mean parameter is the mean of the perturbation, so that the mean of the perturbed variable will be \( \text{constant} + \text{mean} \). Here, the perturbed vector will the sum of

\[
\begin{pmatrix}
-1. \\
1.
\end{pmatrix}
+ \begin{pmatrix}
0.5. \\
0.5.
\end{pmatrix}
\]

and a random vector generated with a centered normal distribution of variance

\[
\begin{pmatrix}
1. & 0. \\
0. & 1.
\end{pmatrix}
\].

The parameter vector ensures that all the random numbers will be between -1. and 1. The correlation parameter correlates the first sub-vector with each of the following. Since here we have only one vector to perturb, there is no correlation to apply.
Chapter 40

Optimization Solver

An optimization solver in Verdandi is a tool to carry out nonlinear optimization. One optimization library is currently interfaced: NLopt.

40.1 NLopt Solver

40.1.1 NLopt Installation

Download the latest version available from the NLopt web site and follow the NLopt Installation instructions.

40.1.2 Using the Class NLoptSolver

NLopt provides algorithms to solve optimization problems. The class NLoptSolver is a Seldon interface to NLopt. It can be used in a Verdandi data assimilation method.

An example of its use in a method can be found in the Four Dimensional Variational (4DVar) method.

40.1.3 Configuration

The main configuration options are described below.

- **algorithm**: the optimization algorithm. NLopt provides gradient-based optimization algorithms and derivative-free optimization algorithms. The name of the optimization algorithm can be:

<table>
<thead>
<tr>
<th>gradient-based optimization algorithms</th>
<th>derivative-free optimization algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD_LBFGS</td>
<td>LN_PRAXIS</td>
</tr>
<tr>
<td>LD_LBFGS_NOCEDAL</td>
<td>LN_COBYLA</td>
</tr>
<tr>
<td>LD_VAR1</td>
<td>LN_BOBYQA</td>
</tr>
<tr>
<td>LD_VAR2</td>
<td>LN_NEWUOA</td>
</tr>
<tr>
<td>LD_TNEWTON</td>
<td>LN_NEWUOA_BOUND</td>
</tr>
<tr>
<td>LD_TNEWTON_RESTART</td>
<td>LN_NELDERMEAD</td>
</tr>
<tr>
<td>LD_TNEWTON_PRECOND</td>
<td>LN_SBPLX</td>
</tr>
<tr>
<td>LD_TNEWTON_PRECOND_R-RESTART</td>
<td>LN_AUGLAG</td>
</tr>
<tr>
<td>LD_MMA</td>
<td>LN_AUGLAG_EQ</td>
</tr>
<tr>
<td>LD_AUGLAG_EQ</td>
<td></td>
</tr>
<tr>
<td>LD_SLSQP</td>
<td></td>
</tr>
</tbody>
</table>
- `parameter_tolerance`: the relative tolerance on the parameters. When the variation of the parameters, after one step of the algorithm, has changed by less than `parameter_tolerance` multiplied by the value of the parameters, the optimization is stopped. If you do not want to use a particular tolerance termination, you can just set that tolerance to zero and it will be ignored.

- `cost_function_tolerance`: the relative tolerance on the cost function. When the variation of the cost function, after one step of the algorithm, has changed by less than `cost_function_tolerance` multiplied by the value of the cost function, the optimization is stopped. If you do not want to use a particular tolerance termination, you can just set that tolerance to zero and it will be ignored.

- `Niteration_max`: the maximum number of cost function evaluations. It is ignored if it is non-positive.

Here is an example derived from the Four Dimensional Variational (4DVar) configuration file.

```yaml
-- Simulation with assimilation using 4D-VAR.
four_dimensional_variational = {

    nlopt = {

        -- Optimization algorithm (LD_VAR1, LD_LBFGS, LD_SLSQP, LD_MMA, -- LD_TNEWTON, ...).
        algorithm = "LD_VAR1",

        -- Relative tolerance on the optimization parameters. If you do not want -- to use a particular tolerance termination, you can just set that -- tolerance to zero and it will be ignored.
        parameter_tolerance = 1.e-4,

        -- Relative tolerance on the cost function.
        cost_function_tolerance = 1.e-4,

        -- Maximum number of function evaluations. Criterion is disabled
        Niteration_max = -1

    }
}
```
Part IX

Developing in Verdandi
Chapter 41

Dependencies

A few libraries are needed, for compilation, configuration files and linear algebra. Before choosing the dependencies, we carried out detailed comparisons of the available solutions:

- **A thorough study of existing libraries for linear algebra** has been carried out to choose Seldon for providing Verdandi with vectors, matrices and related operations.

- **A study of several libraries to manage configuration files** has initially concluded that GetPot should be used to read the configurations. Eventually, we switched to Ops: the configuration files are written in Lua, which makes the configuration files easy to read and write for the beginner, and yet very powerful for advanced users.

- Verdandi provides a high level interface in Python generated via SWIG. The tools to **build a high level interface on top of the C++ source code** have been listed and studied.

Below is the list of dependencies required by Verdandi:

- The software construction tool SCons (version 1.0 or higher) for compilation.

- Python for both SCons and the optional Python interface (see page Python Interface) to the C++ code; the generation of the interface also requires SWIG.

Note that in Verdandi tarball, you will also find:

- The linear algebra library Seldon.

- Lua, a scripting language.

- Ops, a library for reading Lua configuration files.
Chapter 42

Linear Algebra Library

A thorough study of existing libraries for linear algebra has been carried out to choose Seldon.

42.1 Seldon Main Strengths

Seldon is adapted to data assimilation problems in large dimension for its following key features:

- **Availability of dense and sparse matrices, vectors**
  Seldon provides many different matrix and vector structures, and many functions for computations (linear algebra). Among dense matrices, there are specific structures for rectangular matrices, symmetric matrices, hermitian matrices and triangular matrices. Each type includes several formats. E.g., rectangular dense matrices may be stored by rows or by columns; symmetric dense matrices may be stored as rectangular matrices or only upper part of the matrix is stored (this is the packed form of Blas).

- **Interface to BLAS and LAPACK**
  Seldon is interfaced with Blas (levels 1, 2 and 3) and Lapack, except for functions involving banded matrices (since this format is not available for the moment). If Blas is not available to the user, a few alternative functions (same functions written in C++) may be used.

- **Performance**
  The performance of Seldon was evaluated through the Linear Algebra Libraries study.


- **Convenience**
  Seldon is designed to be efficient and convenient, which is notably achieved thanks to template classes. Exception handling and several debug levels were define to ease the "coding" development.

- **Interface to sparse solver libraries**
  For sparse matrices, Seldon is interfaced with direct solvers of MUMPS, SuperLU and UmfPack. There is a bunch of iterative solvers available in Seldon such as Gmres, BiCgSTAB, Qmr, etc. Thanks to templates, these solvers can be used for any type of matrix and preconditioning, not only Seldon matrices. This is very useful when the user does not store the matrix, but is able to perform a matrix-vector product.

- **Portability**
  Seldon is supposed to be fully compliant with the C++ standard. Therefore, it can be compiled by GNU GCC (>=3.0; tested with version 3.2, 3.3, 3.4, 4.0, 4.1, 4.2, 4.3 and 4.4) and by the Intel C++ compiler icc (tested with icc 7.1 and 8.0). No tests were conducted with proprietary compilers under Unix, but the compliance with the C++ standard should ensure portability. Decent versions of Microsoft Visual C++ (i.e., from Visual C++ 2003) compile Seldon.
– Good knowledge of Seldon in Verdandi development team at INRIA

– Compatibility with distributed matrices and vectors

Seldon was interfaced with PETSc which provides distributed vectors and matrices and many functions for computations. PETSc vectors and matrices can be used through the Seldon objects.

42.2 Seldon Matrices and Vectors

The structures manipulated in Verdandi are Seldon vectors and matrices.

Vectors are instances of the class Vector. Class Vector is a template class: Vector<T, Storage, Allocator>, where

– T is the type of the elements to be stored (e.g. double).
– Storage defines how the vector is stored. Storage is equal to VectFull by default for full vectors, you can set it to VectSparse for sparse vectors.
– Finally, Allocator defines the way memory is managed. It is close to STL allocators.

Matrices are instances of the class Matrix. Class Matrix is a template class: Matrix<T, Prop, -Storage, Allocator>, where

– As for vectors, T is the type of the elements to be stored (e.g. double).
– Prop indicates that the matrix has given properties (symmetric, hermitian, positive definite or whatever). This template parameter is never used by Seldon; so the user may define its own properties. Thanks to this template parameter, the user may overload some functions based on the properties of the matrix. Seldon defines two properties: General (default) and Symmetric.
– Storage defines how the matrix is stored. Matrices may be stored in several ways. RowMajor is the default storage.
– Finally, Allocator defines the way memory is managed. It is close to STL allocators.
Chapter 43

Coding Standard

Introduction

Many coding standards may be found on-line: see, for instance, the list provided at http://www.chris-lott.org/resources/cstyle/. In particular, the C++ Coding Standard by Todd Hoff, available at http://www.possibility.com/Cpp/CppCodingStandard.html provided useful guidelines for the present standard.

The general conventions of part 43.1 shall be applied to all computer codes, in any language. C++ conventions follow in part 43.2.

43.1 General Conventions

The following conventions must be applied in all languages.

43.1.1 Development

1. Codes must be written in American English. This includes variable names, file names, comments, . . .
   Justification — A code should be open to anyone. Even if a code is not intended to be shared initially, this may change. Moreover parts of it may be reused in another project.

2. Optimize only the parts of the code that lead to significant overheads.
   In addition, remember that the compilers partially optimize your code.
   Justification — An optimized code is usually less readable than its straightforward version.

3. A code must compile without warnings.
   At development stage, compile your code with the most restrictive compilation options and with all warning messages enabled.
   Justification — It increases portability, and it may help avoiding mistakes.

4. Global variables must be avoided.
   Justification — Units of a code should be independent from their environment so that they may be reused and so that the overall code may be safer.

43.1.2 Names

5. Use explicit and meaningful names.
   You should first consider that the length does not matter. Actually it does, but many programmers tend to use too short and therefore uninformative names. Usually, the wider scope, the longer name.
   Justification — New programmers should be able to read the code without learning the meaning of the variables and the functions in use. Explicit names also save unnecessary comments. Example (in C++):
if (Species.GetName() == target_species)
    output_concentration = Species.Concentration();

is better than:

// Checks whether current species is the target species.
if (sp.GetName() == species)
    // Retrieves the output concentration.
    c = sp.Concentration();

6. **AVOID CONTRACTIONS.** **IF YOU REALLY NEED A CONTRACTION, ONLY USE IT AS A SUFFIX.**
   Accepted contractions are: tmp (temporary), in (input), out (output), obs (observation).  Example:
   value_in.
   **Justification** — Contractions can easily collide (including with a full word) or have meanings depending on
   the context.  Besides, not all developers may understand your contractions, depending on their habits and
   maybe their native language.  For instance, French developers often contract number into nb whereas a
   native English speaker would use num.

7. **UNLESS A MORE EXPLICIT NAME IS FOUND, PREFERABLY USE i FOR AN INDEX.** **AS FOR DIMENSIONS, USE h FOR TIME INDEX, i ALONG X, j ALONG Y AND k ALONG Z.**
   **Justification** — This is a common practice.

8. **AVOID PLURAL FORMS.**
   Even the name of a vector should not be in plural form.  For instance, a vector of observations can be called
   observation, and the i-th observation is observation(i) which is perfectly clear.  If the plural form
   seems to be required, consider appending _list: e.g., location = location_list(i), even if
   location_list is not an instance of the STL class list.
   **Justification** — The use of plural forms makes it difficult to guess or remember the names of variables and
   methods.  It is quickly unclear whether the plural form was used or not for such and such methods.  The
   same is true with a complex object that stores many data sets: should it be named with plural form because
   it contains a list, or should the singular be preferred because it appears as a single block/object? The easiest
   way to avoid the confusion is to avoid plural forms.

9. **A FIXED NUMBER OF XXX SHOULD BE NXXX.**
   Example: Nstep for the number of steps (if it is fixed), Narray for a number of arrays.

10. **THE BOOLEAN METHODS SHOULD HAVE A PREFIX LIKE IS OR HAS.**
    Examples: IsReady, HasObservation, IsEmpty.
    **Justification** — It makes it clear that the method returns a Boolean.  The question that the method answers is
    very clear too.

**43.1.3 Formatting**

**43.1.3.1 Spaces and Parentheses**

11. **PUT ONE BLANK SPACE BEFORE AND ONE BLANK SPACE AFTER THE OPERATORS:** +, -, /, * (MULTIPLICATION), =, +=, -=, *=, /=, |, & |, &&, <, <=, >, >=, ==, !=, <<.
    You might break this rule in inner parens (at a deep level, e.g., some array index like $i+1$ in a complex
    formula).
    **Justification** — It makes the code much more readable.  It is also a very common practice.

12. **PUT ONE BLANK SPACE AFTER EACH COMMA.**
    **Justification** — It makes the code much more readable.  It is a very common practice.

13. **DO NOT ADD TRAILING SPACES AT THE END OF CODE LINES.**
    A script can remove the trailing spaces for you—such a script should be run before any commit to the repository
    of your revision control system.  Emacs users may have their editor removing the trailing spaces whenever
    they save a file.  They can also have Emacs show them the trailing whitespaces; e.g., in Python mode:

    (setq whitespace-style '((trailing))
    (add-hook 'python-mode-hook 'whitespace-mode)

Generated on Mon Sep 28 2015 16:49:58 for Verdandi by Doxygen
Justification — Browsing the code, moving blocks, copies, … are slowed down because of trailing spaces. In addition, the differences between two revisions of a file should not include such noise.

14. **NO SPACE BETWEEN THE FUNCTION NAME AND ITS ARGUMENTS LIST.**
   Write
   
   `func(a, b)`
   
   instead of
   
   `func(a, b)`

15. **NO SPACE AFTER AN OPENING PAREN OR A CLOSING PAREN.**
    Write
    
    `func(a, b)`
    
    instead of
    
    `func(a, b)`

16. **PUT A SPACE BETWEEN A LANGUAGE KEYWORD AND THE FOLLOWING PAREN.**
    Write
    
    `while (error > epsilon)`
    
    instead of
    
    `while(error > epsilon)`

    *Justification* — Keywords and functions should be distinguishable.

17. **DO NOT PUT UNNECESSARY PARENS IN LOGICAL EXPRESSIONS, EXCEPT TO CLARIFY THE ORDER OF EVALUATION.**
    For instance (in C++), write
    
    `if (i != 0 && j > 5)`
    
    instead of
    
    `if ((i != 0) && (j > 5))`

    *Justification* — Unnecessary parens slow down the reading.

### Comments

18. **A COMMENT LINE IS PLACED BEFORE THE LINES OR THE BLOCK IT COMMENTS.**
    Observe where each comment line is placed:
    
    ```
    // Checks the availability of observations. Note that the call implicitly loads the observations at current date.
    if (observation_manager.HasObservation())
      // Assimilates the available observations.
      
      Analyze();
      if (positive_state)
        // Enforces the positivity of the state vector.
        for (int i = 0; i < Nstate; i++)
          state(i) = max(0., state(i));
    ```
Justification — Since an explanation may address several lines, the scope of a comment placed after is unclear.

19. **LIKE A SENTENCE, A COMMENT STARTS WITH A CAPITAL AND ENDS WITH A DOT (EVEN A ONE-WORD COMMENT).**
   *Justification* — All comment lines should be consistent. This rule is clear and easy to follow.

20. **USE SIMPLE PRESENT TO EXPLAIN WHAT A LINE DOES OR WHAT A SEQUENCE OF LINES DOES.**
    **See the example above.**
    *Justification* — A comment introduces to what a line does. So // Extracts data. implicitly means // This line extracts data.

21. **WHEN REFERRING TO A VARIABLE, SURROUND THE VARIABLE NAME WITH SIMPLE QUOTES.**
    **Example:** // Updates ‘state’ so that it should be consistent with ‘full_state’.
    *Justification* — Reading comments, with variable names included, can be really difficult because the variable names are often words: one cannot identify at first sight that the variable name is a special element. Think of a comment line like // Makes a consistent with the location. instead of // Makes ‘a’ consistent with the location.

43.1.3.3 Function and Method Definition

22. **ARGUMENTS ARE SORTED FROM INPUT VARIABLES TO OUTPUT VARIABLES. DIMENSIONS ARE PROVIDED FIRST.**
    The only exception is for optional arguments if they are necessarily the last arguments (as in C++ and Python).
    Try to sort all arguments so that the order makes sense. For instance, if the input variables are a number of points along \( x \), the abscissae and the values of a function \( f \) at these abscissae, then provide them in that order. Indeed, one needs first the number of points \( n \), then the positions \( x_i \) of the points \( i \in [0,n-1] \) and finally the associated values \( f(x_i) \).
    *Justification* — A code line is read from left to right, and obviously an input comes before an output.

43.1.4 Language Features

43.1.4.1 Standard

23. **BUILD A FULLY STANDARD-COMPLIANT CODE. IF A COMPILER DOES NOT UNDERSTAND IT, DISCARD IT OR MAYBE MAINTAIN SPECIFIC CODE FOR IT.**
    *Justification* — This ensures portability and makes the code perennial.

43.2 C++

43.2.1 Development

24. **COMPILATION WITH GNU G++ AND WITH OPTIONS Wall -ansi -pedantic SHOULD NOT ISSUE ANY WARNING.**
    *Justification* — It increases portability (through compliance with the C++ standard), and it may help avoiding mistakes.

43.2.2 Names

43.2.2.1 Common Conventions

25. **DO NOT ADD A PREFIX TO A SET OF OBJECTS TO AVOID CONFLICTS.**
    Use name spaces instead.
43.2.2.2 Classes and Methods

26. A NAME IS ONE WORD OR A CONCATENATION OF WORDS. THE FIRST LETTER OF EACH WORD (INCLUDING THE FIRST ONE) IS UPPERCASE.

Examples:

    class FormatBinary;
    double Data::GetMax() const;
    void Data::Print() const;
    void LoadPreviousAnalysis(...);

**Justification** — Two other conventions are widely used: formatBinary and format_binary. With the latter convention, classes and methods are not easily identified in a code. The former convention is a bit inconsistent: one-word methods are not as emphasized as two-word methods are.

27. THE ACCESSORS SHOULD BE NAMED WITH Get OR Set (PREFIXED).

Example: GetDate, SetPosition.

43.2.2.3 Functions

28. SAME RULES AS FOR THE METHODS.

Meanwhile, the name of a small function (e.g., a single formula, or an extension of the C++ libraries) may be lowercase with underscores to delimit the words.

29. EXTERN FUNCTIONS SHOULD BE LOWERCASE WITH UNDERSCORES TO DELIMIT THE WORDS, AND THEY SHOULD BE PREFIXED BY AN UNDERSCORE.

The name of an extern function is defined by the compiler. If the compiler does comply with this convention, define a macro that follows this convention. Example:

    #define _linear_interpolation linear_interpolation_

**Justification** — Fortran functions are usually named with one or two underscores at the end. The rule is consistent with the addition of an underscore, but as a prefix in order to avoid conflicts (with attributes, see below).

43.2.2.4 (Local) Variables

This section also applies to method arguments and function arguments.

30. USE LOWER CASE AND WORDS DELIMITED WITH UNDERSCORES.

**Justification** — Many variables are naturally lowercase, like the indexes. In addition, one may declare an instance of a class, say Data or ObservationManager, with the same name as the class: Data data or ObservationManager observation_manager.

43.2.2.5 Attributes

31. SAME RULES AS FOR VARIABLES, EXCEPT THAT AN UNDERSCORE MUST BE APPENDED AT THE END OF THE NAME.

This rule might be broken in case the attribute is public or for consistency with a public attribute that has no appended underscore.

**Justification** — The underscore at the end enables to distinguish attributes from local variables within the methods. The scope is an important property of a variable. In addition, the method arguments may have the same names as the attributes, without the underscore. For example:

    void ExtendedStream::SetDelimiter(delimiter)
    {
        delimiter_ = delimiter;
    }
43.2.2.6 References, Pointers, Global Variables and Constant Variables

32. **No special notation is associated with references, pointers, global variables or constant variables.**

   *Justification* — Constant variables are declared as such (keyword `const`); no alteration of these variables can occur. Global variables should be avoided. References are used in C++ to manipulate variables just like others: a notation to distinguish them would break this advantage. Programmers sometimes prefix a ‘p’ for pointers, but the syntax is usually clear enough to show that a pointer is in use.

43.2.2.7 Name Spaces

33. **Namespaces are mainly used for libraries. A namespace has the exact name of its library.**

   Example:
   ```
   namespace Verdandi;
   ```

43.2.2.8 Type Names (typedef)

34. **Use lower case and words delimited with underscores. Do not append an underscore even if the type name is defined in a class.**

   *Justification* — Type names are used as shortcuts for what may be seen as a low-level type, just like a string or a double in the `main()` function.

43.2.2.9 Macros

35. **Use upper case and words delimited by underscores.**

   *Justification* — Macros should be clearly identified because of their really specific nature. In addition, this is common practice.

36. **In case a macro is related to a library, the first word of the macro must be the library name.**

   Example:
   ```
   #define VERDANDI_DEBUG_LEVEL_4
   ```

   *Justification* — This avoids conflicts with macros from other libraries, and it better indicates what the macro is for.

43.2.3 Formatting

43.2.3.1 Indentation and Braces

37. **Use the Allman standard for indentation.**

   This indentation style is called BSD under Emacs. It looks like this:

   ```
   if (i == f(a, b))
       i += 5; // No braces for a single line.
   else
       { // Instead of "else {".
       while (i != 5)
           { // Needed, because "}" is missing.
             j = 3 * f(4, b);
             ++i;
           }
       i--; // Some people still like to put it before the end of a code block.
   }
   ```
Note the (compulsory) 4-space depth for the indentation. Emacs users can enforce this convention with the following code (placed in .emacs):

```lisp
(defun verdandi-c++-mode ()
  (interactive)
  (c-set-style "bsd")
  (setq c-basic-offset 4)
  (add-hook 'before-save-hook 'delete-trailing-whitespace t t))

(defun verdandi-c++-mode-hook ()
  (if (string-match "verdandi" buffer-file-name)
      (verdandi-c++-mode)))

(add-hook 'c++-mode-hook 'verdandi-c++-mode-hook)
```

The hook `delete-trailing-whitespace` will remove trailing spaces when the file is saved, which is another formatting rule (see section 43.1.3.1). The Verdandi C++ mode will be applied to any file that contains the word “verdandi” in its absolute path.

**Justification** — Consistent indentation is utterly required at least to browse the code and to grasp its structure. In addition, differences between two versions of a same code are easier to parse with a standard indentation.

38. **Tabulations are forbidden. They should be replaced with spaces.**

Emacs users can add the following line in their .emacs:

```lisp
(setq-default indent-tabs-mode nil)
```

**Justification** — Tabulation-based indentation may be more convenient for code browsing than space-based indentation: moving in the code may be faster. Unfortunately, the length of a tabulation may vary from one environment to another (usual values are 4 or 8 spaces, maybe 2 sometimes). This can cause misalignment. Take for example:

```c
<TAB>if (condition)
<TAB><TAB>long_function_name(a, b, c,
<TAB><TAB><TAB><TAB>....h, j, i);
```

where `<TAB>` is a tabulation and `. ` is a whitespace. If the tabulation length is decreased, the code may look misaligned:

```c
<TAB><TAB>if (condition)
<TAB><TAB>long_function_name(a, b, c,
<TAB><TAB><TAB><TAB>....h, j, i);
```

where h, j, i is not properly placed. A clever solution is to use tabulations for block indentation and to use spaces for alignment:

```c
<TAB>if (condition)
<TAB><TAB>long_function_name(a, b, c,
<TAB><TAB><TAB><TAB>.........h, j, i);
```

In that configuration, the formatting will be fine whatever the tabulation length. Unfortunately, it seems that only Emacs and vi implement that formatting. In addition, the interaction between such a convention and the limit on the line length may be an issue. To conclude, the safest indentation strategy is to use spaces only, which any decent editor should handle:

```c
....if (condition)
.........long_function_name(a, b, c,
.................h, j, i);
```

39. **The indentation depth is four spaces.**

**Justification** — The indentation depth is usually four spaces (like in GNU coding standards) or eight spaces (like in Linux kernel). In scientific computing, several indentation levels are commonly required—for example for loops in three-dimensional space. A eight-space indentation depth is not practical in that context.

40. **Do not put a useless semi-colon at the end of a block.**

Only classes and structures declarations require a semi-colon after the closing brace.
43.2.3.2 Comments and Documentation

41. **Use //, not /*, except for Doxygen comments.**

42. **Include Doxygen comments for every class, every attribute, every method and every function.**

43. **With respect to Doxygen comments of functions and methods:**
   - There must be a brief description, introduced by //!, or \\!brief if the description does not fit into a single line;
   - There must be a description for every argument and for the returned value:
     - Only use \param[in], \param[in,out] and \param[out] to introduce the description for an argument;
     - The description for an argument starts lowercase and ends with a dot;
     - The description for a returned value starts with a capitalized word and ends with a dot.
   - A full description should also be added whenever necessary, just before the arguments description;
   - Any reference to an argument, say \texttt{x}, should be introduced with the Doxygen command \a:x.

Template:

```
//! Here comes the brief description.
/*! Here comes the long description, if needed.
\param[in] \texttt{x} description of the first parameter. It may include several sentences on several lines.
\param[in,out] value another parameter.
\param[out] error \texttt{true if and only if an error occurred.}
*/
```

Another template:

```
/*! \brief Here goes a brief description that does not fit into a single line. */
/*! Here comes the long description, if any.
\param[in] value description of the argument.
\return The sign of \texttt{a} value.
*/
```

**Justification** — These rules ensure that the Doxygen documentation is complete and clean.

44. **Doxygen comments for functions and methods are put before the definition in the source file, not before the declaration in the header file.**

**Justification** — Headers should be as light as possible so that one may browse them quickly.

45. **Doxygen comments for classes and attributes are put before their declarations, in the header file.**

For attributes, the description is introduced by //!, or \\!brief if it does not fit into a single line.

**Justification** — There is no other suitable place.

46. **The code may be organized in sections and subsections that are introduced as follows.**

Section:

```
<two blank lines>
////////////////////////////////////
// READS INPUT FIELDS //
////////////////////////////////////
<two blank lines>
```
Long section (especially in libraries, with many functions in the section—otherwise prefer the previous format):

```cpp
namespace AtmoData
{
    // code
}
```

Heavy subsection:

```cpp
/*****************
* Binary files *
*****************/
```

Light subsection (prefer this to the heavy subsection, except if the subsection is long—you may then nest light subsections inside the heavy subsection):

```cpp
/*** Binary files ***/
```

47. NAME SPACES ARE INSERTED THIS WAY:

```cpp
namespace AtmoData
{
    // code
}
```

43.2.3.3 Lines

48. ONLY ONE STATEMENT SHOULD BE PUT ON A SINGLE LINE.

```cpp
int j;
string line;
if (i == 5)
    cout << "Done." << endl;
```

```cpp
int j; string line;
if (i == 5) cout << "Done." << endl;
```

There should never be two semi-colons on the same line, except in for statements:

```cpp
for (i = 0; i < 10; i++)
    cout << i << endl;
```

**Justification** — This usually makes the code clearer. It avoids misleading implementations like:
if (i == 5)
  cout << "Done." << endl; i++;

49. FUNCTIONS AND METHODS DEFINITIONS SHOULD BE SEPARATED BY TWO BLANK LINES.
   Justification — There should be more space between two functions than between two blocks in a function.

50. A LINE SHOULD NOT CONTAIN STRICTLY MORE THAN 78 CHARACTERS.
   Justification — This ensures that the code may be properly printed and that it could be displayed on a screen
   in text mode. Furthermore, one often needs to display two source files side by side. The differences between
   two files can also be displayed side by side.

51. IF YOU NEED TO BREAK A LINE THAT INTRODUCES THE DEFINITION OF A METHOD, NEVER SEPARATE THE TWO
    COLONS :: FROM THE NAME OF THE METHOD.
    Write
    ClassName
    ::MethodName(...long argument list...)

   instead

    ClassName::
    MethodName(...long argument list...)

   Justification — One often searches for the definition of a given method in a source file. With this rule, one
   can search for ::MethodName (using the search ability of one’s text editor) to quickly find the definition. A
   search for MethodName may be inefficient because there may be many calls to the method, at different
   places.

43.2.3.4 Variable Definition

52. DO NOT SYSTEMATICALLY DECLARE ALL VARIABLES AT THE BEGINNING OF A PROGRAM OR A FUNCTION.
    DECLARE SMALL BUNCHES OF VARIABLES INSTEAD.
    Old languages require that all variables are declared at the very beginning. This is the reason why this
    convention is still in use, even in modern programming languages. It is good idea to declare at the beginning
    a few variables that will be used at many places in the current block, like an index i. Otherwise, declare the
    variables more locally.
    Justification — A variable declaration should be close to the lines where it is used so that the programmer
    can easily access to this declaration and so that the scope of the variable can be restricted.

53. CHARACTERS * AND & SHOULD BE DIRECTLY CONNECTED TO THE TYPE, NOT THE VARIABLE.
    Write
    int &i;

   instead of

    int &i;

   Justification — In the previous example, the type of i is int&, and it should appear as such.

54. CONSTANTS ARE DECLARED WITH a const STATEMENT, NOT WITH #define OR SO.

43.2.3.5 Class Definition

55. ALL ATTRIBUTES SHOULD BE PROTECTED {protected}.
    Public attributes are a bad idea because there are really unsafe. If there are so many attributes that maintain-
    ing accessors is difficult, this rule might be broken. Private attributes can be fine, but they cannot be accessed
    by the derived classes, which is rarely a useful feature.
    Justification — Attributes should be accessed and modified only with methods, so as to allow miscellaneous
    checks, and so as to guaranty at any time the consistency of the object.
56. **In a class definition, put, in this order:**

(A) The `typedef` declarations;
(B) The attributes;
(C) The constructor(s);
(D) The destructor;
(E) The public methods;
(F) The protected methods;
(G) The private methods.

In the source file, the definitions must follow the same order as the declarations in the header file.

57. **Provide constant methods** (`const`) **whenever possible.**
   *Justification* — It makes the code safer, and such methods are compulsory to manipulate `const` objects.

58. **Declare a virtual destructor in case the class is derived and contains virtual methods.**
   *Justification* — If a pointer to an instance of a derived class is used, then `delete` will only call the base destructor. In addition, some compilers will issue a warning.

### 43.2.3.6 Another Rule

59. **Floating-point numbers should always have a decimal point.**
   While manipulating floating-point numbers, write

   ```cpp
   double x = 2.; // or 2.0
   double y = 2. * x;
   y = 1. + x / 3.;
   ```

   instead of

   ```cpp
   double x = 2;
   double y = 2 * x;
   y = 1 + x / 3;
   ```

   *Justification* — It clearly shows what type of variable is manipulated. It can avoid mistakes like writing `2 / 3`
   (which is zero) instead of `2. / 3.` (which is of course not zero).

### 43.2.4 Files

#### 43.2.4.1 General Rule

60. **Definition shall never follow declaration. Put declarations in header files and definitions in source files. There must be a header file for any source file.**
   *Justification* — First, a precompiled library can be built only if declarations and definitions have been split.
   Second, the contents of a library may be quickly browsed in its headers.

#### 43.2.4.2 Names

61. **Extensions are `hpp` or `hxx` (for headers) and `cpp` or `cxx` (for sources). `*.xx` files should be used for libraries, exclusively.**
   *Justification* — It is convenient to identify what is part of a software (to be compiled) and what is part of a library (to be included).

62. **For libraries, a header file and a source file should be associated to each class. Those files have the same name as the class and they match its case.**
63. **The names of the directories and the names of the files to be compiled are lowercase with underscores to delimit the words.**
   Lower case is used for the files not part of the core library (*.cpp): examples, unit tests, ...
   *Justification* — Browsing the code from command line is easier with lowercase directory names, and it is consistent with Linux/Unix conventions. A file name of the core library should be the same as the class it implements (see above), but other files should be lowercase also for convenience and consistency with the environment.

43.2.4.3 **Includes**

64. **All library files must have include guards.**

65. **Include guards must be in the form {LIBRARY NAME (UPPER CASE)}_FILE_{(FILE NAME WITH ITS FULL PATH IN THE LIBRARY (UPPER CASE; SLASHES AND THE DOT ARE REPLACED WITH AN UNDERSCORE)}.**
   Example: SELDON_FILE_SHARE_VIRTUALALLOCATOR_HXX for the file “VirtualAllocator.hxx” in directory “share” of the library Seldon.

66. **Include libraries of the C++ standard with < and >, and other libraries with double quotes.**
   Example:
   ```
   #include <vector>
   #include "Seldon.hxx"
   ```

43.2.5 **About C++ Features**

43.2.5.1 **Exceptions**

67. **Use exceptions to manage errors.**
   *Justification* — First, exception are in C++ precisely to manage errors. In old languages, one adds an integer to the arguments of all functions in order to track errors. This is difficult to maintain and it is dangerous because errors may be detected without further action. The other practice is to simply terminate the program (e.g., with abort()), but then the program cannot recover from the error.

68. **Do not use exception specifications.**
   *Justification* — It is a nightmare to keep these exception specifications up-to-date because of exceptions that may be thrown by nested functions.

69. **Check every memory allocation.**
   This is usually managed at a rather low level, in the libraries that provide the base structures.
   *Justification* — If a memory allocation fails, a program should not keep running.

70. **Allow the user to mute error checking.**
   To achieve this, enclose any test and its throw statement by #ifdef (library name (upper case))_DEBUG_(small description) and #endif. *Example: VERDANDI_DEBUG_CHECK_DIMENSION*
   *Justification* — Some tests may result in significant overheads. For instance, in the access to an element of a vector, checking the validity of the index requires a significant amount of time compared to the access itself. The test may be very helpful, but one should be able to deactivate it.

43.2.5.2 **Templates**

71. **For scientific computing, always consider template functions: most functions should have the numerical type as template parameter.**
   Do not fear templates! They are really helpful to build generic codes while maintaining high performance.
   *Justification* — In scientific computing, the type of the underlying data may change: float, double, complex<float>, complex<double> or even a user-defined class. In addition, one never perfectly predicts what will be the eventual use of one’s code; for instance, parts of it may be reused in another context, with other data structures.
43.2.5.3 C++ Standard

72. **Build a fully standard-compliant code.** If a compiler does not understand it, discard it or write the code it needs within a `#define/#endif` block.
   *Justification* — This ensures portability and makes the code perennial.

73. **Do not use C features if they have been replaced by C++ features.**
   Even if C++ features do not always seem better at first sight, trust the designers of the C++ standard.
   *Justification* — C++ features are better than their C equivalents, except that they might lead to overheads in a few cases.

43.2.6 Other Conventions

74. **In a loop, the stopping test should use an inclusive lower-bound or an exclusive upper-bound.**
   Example:
   ```
   for (unsigned int i = 0; i < 50; i++)
   ```

43.3 Python

75. **Tabulations are forbidden. They should be replaced with spaces.**
   *Justification* — In Python, the indentation is part of the language since it defines the blocks. If the indentation changes because of a varying tabulation length and a mixing of spaces and tabulations, the code changes. This is a risk nobody is willing to take.

76. **The indentation depth is four spaces.**
   *Justification* — Same as in C++, see section 43.2.3.1.

77. **A line should not contain strictly more than 78 characters.**
   *Justification* — Same as in C++, see section 43.2.3.3.

78. **Use Doxygen, with similar to conventions to those for C++—section 43.2.3.2.**

This part of the document should be completed later. Note that the conventions should be similar to C++. The “Style Guide for Python Code”, by Guido van Rossum and Barry Warsaw, [http://www.python.org/dev/peps/pep-0008/](http://www.python.org/dev/peps/pep-0008/), should serve as a sound background.

43.4 Other Languages

79. **Only use a limited number of languages.**
   For instance, with C++ and Python, one can cover all needs: from low-level high-performance computing to high level programming (scripts, command line).
   *Justification* — It is better to fully understand a few languages than poorly using many languages. Moreover adding dependencies (to other languages and, as a consequence, to other compilers and libraries) may decrease the portability and may increase the installation difficulty.

80. **Try to follow the rules associated with C++.**
   In case you really need to use another language, C++ conventions should cover most features of this additional language.
Part X

Reference Documentation
Chapter 44

Namespace Index

44.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Verdandi
   The namespace of the data assimilation library Verdandi ........................ 169
## Chapter 45

### Hierarchical Index

#### 45.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

<table>
<thead>
<tr>
<th>Class Type</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verdandi::BalgovindMatrix</td>
<td>185</td>
</tr>
<tr>
<td>Verdandi::BaseForecaster</td>
<td>188</td>
</tr>
<tr>
<td>Verdandi::BaseForecaster&lt; T &gt;</td>
<td>188</td>
</tr>
<tr>
<td>Verdandi::DiscountedRidgeRegression</td>
<td>220</td>
</tr>
<tr>
<td>Verdandi::BasePerturbationManager&lt; NewranPerturbationManager &gt;</td>
<td>195</td>
</tr>
<tr>
<td>Verdandi::NewranPerturbationManager</td>
<td>340</td>
</tr>
<tr>
<td>Verdandi::BasePerturbationManager&lt; RandomPerturbationManager &gt;</td>
<td>195</td>
</tr>
<tr>
<td>Verdandi::RandomPerturbationManager</td>
<td>??</td>
</tr>
<tr>
<td>Verdandi::BasePerturbationManager&lt; TR1PerturbationManager &gt;</td>
<td>195</td>
</tr>
<tr>
<td>Verdandi::TR1PerturbationManager</td>
<td>477</td>
</tr>
<tr>
<td>Verdandi::BasePerturbationManager&lt; TRNGPerturbationManager &gt;</td>
<td>195</td>
</tr>
<tr>
<td>Verdandi::TRNGPerturbationManager</td>
<td>484</td>
</tr>
<tr>
<td>ClassTest</td>
<td>217</td>
</tr>
<tr>
<td>Verdandi::DiagonalMatrix</td>
<td>218</td>
</tr>
<tr>
<td>Verdandi::Error</td>
<td>231</td>
</tr>
<tr>
<td>Verdandi::ErrorArgument</td>
<td>233</td>
</tr>
<tr>
<td>Verdandi::ErrorConfiguration</td>
<td>234</td>
</tr>
<tr>
<td>Verdandi::ErrorIO</td>
<td>236</td>
</tr>
<tr>
<td>Verdandi::ErrorProcessing</td>
<td>238</td>
</tr>
<tr>
<td>Verdandi::ErrorPythonUndefined</td>
<td>239</td>
</tr>
<tr>
<td>Verdandi::ErrorUndefined</td>
<td>241</td>
</tr>
<tr>
<td>Verdandi::IsotropicBalgovindMatrix</td>
<td>283</td>
</tr>
<tr>
<td>LinearObservationManager::LinearObservationManager</td>
<td>310</td>
</tr>
<tr>
<td>Verdandi::Logger</td>
<td>315</td>
</tr>
<tr>
<td>Verdandi::MessageHandler</td>
<td>324</td>
</tr>
<tr>
<td>Verdandi::ObservationAggregator</td>
<td>345</td>
</tr>
<tr>
<td>Verdandi::OutputSaver</td>
<td>364</td>
</tr>
<tr>
<td>PythonModelTemplate::PythonModelTemplate</td>
<td>413</td>
</tr>
<tr>
<td>PythonObservationManagerTemplate::PythonObservationManagerTemplate</td>
<td>428</td>
</tr>
<tr>
<td>QuadraticModel::QuadraticModel</td>
<td>444</td>
</tr>
<tr>
<td>Verdandi::TrajectoryManager</td>
<td>482</td>
</tr>
<tr>
<td>Verdandi::Variable</td>
<td>493</td>
</tr>
<tr>
<td>Verdandi::VerdandiBase</td>
<td>495</td>
</tr>
<tr>
<td>Verdandi::BasePerturbationManager</td>
<td>195</td>
</tr>
<tr>
<td>Verdandi::CheckingModel</td>
<td>200</td>
</tr>
<tr>
<td>Verdandi::ClampedBar</td>
<td>207</td>
</tr>
<tr>
<td>Class Name</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Verdandi::EnsembleKalmanFilter</td>
<td>225</td>
</tr>
<tr>
<td>Verdandi::ExtendedKalmanFilter</td>
<td>243</td>
</tr>
<tr>
<td>Verdandi::ExtendedMinimaxFilter</td>
<td>??</td>
</tr>
<tr>
<td>Verdandi::ForwardDriver</td>
<td>247</td>
</tr>
<tr>
<td>Verdandi::FourDimensionalVariational</td>
<td>250</td>
</tr>
<tr>
<td>Verdandi::FrontPositionObserver</td>
<td>??</td>
</tr>
<tr>
<td>Verdandi::GridToNetworkObservationManager</td>
<td>254</td>
</tr>
<tr>
<td>Verdandi::HamiltonJacobiBellman</td>
<td>277</td>
</tr>
<tr>
<td>Verdandi::LevelSetObservationManager</td>
<td>??</td>
</tr>
<tr>
<td>Verdandi::LinearObservationManager</td>
<td>286</td>
</tr>
<tr>
<td>Verdandi::Lorenz</td>
<td>318</td>
</tr>
<tr>
<td>Verdandi::ModelTemplate</td>
<td>325</td>
</tr>
<tr>
<td>Verdandi::MonteCarlo</td>
<td>335</td>
</tr>
<tr>
<td>Verdandi::Nudging</td>
<td>??</td>
</tr>
<tr>
<td>Verdandi::ObservationGenerator</td>
<td>351</td>
</tr>
<tr>
<td>Verdandi::ObservationManagerTemplate</td>
<td>354</td>
</tr>
<tr>
<td>Verdandi::OptimalInterpolation</td>
<td>360</td>
</tr>
<tr>
<td>Verdandi::PetscClampedBar</td>
<td>368</td>
</tr>
<tr>
<td>Verdandi::PetscLinearObservationManager</td>
<td>378</td>
</tr>
<tr>
<td>Verdandi::PythonModel</td>
<td>402</td>
</tr>
<tr>
<td>Verdandi::PythonObservationManager</td>
<td>422</td>
</tr>
<tr>
<td>Verdandi::QuadraticModel</td>
<td>432</td>
</tr>
<tr>
<td>Verdandi::ReducedMinimax</td>
<td>450</td>
</tr>
<tr>
<td>Verdandi::ReducedOrderExtendedKalmanFilter</td>
<td>456</td>
</tr>
<tr>
<td>Verdandi::ReducedOrderUnscentedKalmanFilter</td>
<td>460</td>
</tr>
<tr>
<td>Verdandi::ShallowWater</td>
<td>465</td>
</tr>
<tr>
<td>Verdandi::UnscentedKalmanFilter</td>
<td>488</td>
</tr>
<tr>
<td>Verdandi::VerdandiOps</td>
<td>497</td>
</tr>
</tbody>
</table>
Chapter 46

Class Index

46.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Verdandi::BalgovindMatrix
   This class defines a covariance matrix in Balgovind form ........................................ 185
Verdandi::BaseForecaster ................................................................. 188
Verdandi::BasePerturbationManager
   This class generates and applies perturbations ......................................................... 195
Verdandi::CheckingModel
   This class is a model template ................................................................................. 200
Verdandi::ClampedBar
   This class is a clamped-bar model ............................................................................ 207
ClassTest
   This class is a test class to run an example .............................................................. 217
Verdandi::DiagonalMatrix
   Diagonal covariance error matrix ............................................................................. 218
Verdandi::DiscountedRidgeRegression ................................................................. 220
Verdandi::EnsembleKalmanFilter
   This class implements the ensemble Kalman filter ..................................................... 225
Verdandi::Error
   This class serves for exceptions raised when an error is found .................................. 231
Verdandi::ErrorArgument
   This class serves for exceptions raised when a function or a method is called with an erroneous argument .......................................................................................................... 233
Verdandi::ErrorConfiguration
   This class serves for exceptions raised when an error is found in a configuration ......... 234
Verdandi::ErrorIO
   This class serves for exceptions raised when an input/output error occurs .................... 236
Verdandi::ErrorProcessing
   This class serves for exceptions raised when an error occurs during some data processing . 238
Verdandi::ErrorPythonUndefined
   This class serves for exceptions raised when an undefined function or method in a Python module is called ..................................................................................................................... 239
Verdandi::ErrorUndefined
   This class serves for exceptions raised when an undefined function or method is called ................................................................. 241
Verdandi::ExtendedKalmanFilter
   This class implements the extended Kalman filter .................................................... 243
Verdandi::ExtendedMinimaxFilter
   This class implements the extended Kalman filter ...................................................... ??
Verdandi::ForwardDriver
   This class simply performs a forward simulation ...................................................... 247
<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verdandi::FourDimensionalVariational</td>
<td>This class implements 4D-Var</td>
<td>250</td>
</tr>
<tr>
<td>Verdandi::FrontPositionObserver</td>
<td>This class simply performs a forward simulation</td>
<td>??</td>
</tr>
<tr>
<td>Verdandi::GridToNetworkObservationManager</td>
<td>Observation operator that maps from a grid to a network</td>
<td>254</td>
</tr>
<tr>
<td>Verdandi::HamiltonJacobiBellman</td>
<td>This class is a solver for Hamilton-Jacobi-Bellman equation</td>
<td>277</td>
</tr>
<tr>
<td>Verdandi::IsotropicBalgovindMatrix</td>
<td>This class defines a covariance matrix in isotropic Balgovind form</td>
<td>283</td>
</tr>
<tr>
<td>Verdandi::LevelSetObservationManager</td>
<td>This class is a template of observation manager</td>
<td>??</td>
</tr>
<tr>
<td>LinearObservationManager::LinearObservationManager</td>
<td>This class is a linear observation manager written in Python</td>
<td>310</td>
</tr>
<tr>
<td>Verdandi::LinearObservationManager</td>
<td>Linear observation operator</td>
<td>286</td>
</tr>
<tr>
<td>Verdandi::Logger</td>
<td>Logging class</td>
<td>315</td>
</tr>
<tr>
<td>Verdandi::Lorenz</td>
<td>This class is a Lorenz model</td>
<td>318</td>
</tr>
<tr>
<td>Verdandi::MessageHandler</td>
<td>This class enables objects to communicate using messages</td>
<td>324</td>
</tr>
<tr>
<td>Verdandi::ModelTemplate</td>
<td>This class is a model template</td>
<td>325</td>
</tr>
<tr>
<td>Verdandi::MonteCarlo</td>
<td>This class performs allows to perform Monte Carlo simulations</td>
<td>335</td>
</tr>
<tr>
<td>Verdandi::NewranPerturbationManager</td>
<td>This class generates random samples using Newran</td>
<td>340</td>
</tr>
<tr>
<td>Verdandi::Nudging</td>
<td></td>
<td>??</td>
</tr>
<tr>
<td>Verdandi::ObservationAggregator</td>
<td>Observation manager which computes an aggregated observation vector</td>
<td>345</td>
</tr>
<tr>
<td>Verdandi::ObservationGenerator</td>
<td>This class simply performs a forward simulation</td>
<td>351</td>
</tr>
<tr>
<td>Verdandi::ObservationManagerTemplate</td>
<td>This class is a template of observation manager</td>
<td>354</td>
</tr>
<tr>
<td>Verdandi::OptimalInterpolation</td>
<td>This class performs optimal interpolation</td>
<td>360</td>
</tr>
<tr>
<td>Verdandi::OutputSaver</td>
<td>This class provides convenient methods to save variables on disk</td>
<td>364</td>
</tr>
<tr>
<td>Verdandi::PetscClampedBar</td>
<td>This class is a clamped-bar model</td>
<td>368</td>
</tr>
<tr>
<td>Verdandi::PetscLinearObservationManager</td>
<td>Linear observation operator</td>
<td>378</td>
</tr>
<tr>
<td>Verdandi::PythonModel</td>
<td></td>
<td>402</td>
</tr>
<tr>
<td>PythonModelTemplate::PythonModelTemplate</td>
<td>This class is a template model in Python</td>
<td>413</td>
</tr>
<tr>
<td>Verdandi::PythonObservationManager</td>
<td>This class is an interface for Python observation managers</td>
<td>422</td>
</tr>
<tr>
<td>PythonObservationManagerTemplate::PythonObservationManagerTemplate</td>
<td>This class is a template observation manager in Python</td>
<td>428</td>
</tr>
<tr>
<td>QuadraticModel::QuadraticModel</td>
<td>This class is a quadratic model written in Python</td>
<td>444</td>
</tr>
<tr>
<td>Verdandi::QuadraticModel</td>
<td>This class is a quadratic model</td>
<td>432</td>
</tr>
<tr>
<td>Verdandi::RandomPerturbationManager</td>
<td>This class generates random samples using the random library from C++</td>
<td>??</td>
</tr>
<tr>
<td>Verdandi::ReducedMinimax</td>
<td>This class implements a reduced minimax filter</td>
<td>450</td>
</tr>
</tbody>
</table>
Verdandi::ReducedOrderExtendedKalmanFilter
   This class implements a reduced order extended Kalman filter 456
Verdandi::ReducedOrderUnscentedKalmanFilter
   This class implements a reduced order unscented Kalman filter 460
Verdandi::ShallowWater
   This class is a shallow-water model 465
Verdandi::TR1PerturbationManager
   This class generates random samples using C++ TR1 library 477
Verdandi::TrajectoryManager
   This class manages model trajectory 482
Verdandi::TRNGPerturbationManager
   This class generates random samples using TRNG 484
Verdandi::UnscentedKalmanFilter
   This class implements the unscented Kalman filter 488
Verdandi::Variable
   This class enables to define a variable for the output saver 493
Verdandi::VerdandiBase
   Base class for Verdandi objects 495
Verdandi::VerdandiOps
   This class extends the Ops::Ops class with Verdandi-related features 497
Chapter 47

Namespace Documentation

47.1 Verdandi Namespace Reference

The namespace of the data assimilation library Verdandi.

Classes

- class BalgovindMatrix
  This class defines a covariance matrix in Balgovind form.
- class DiagonalMatrix
  Diagonal covariance error matrix.
- class IsotropicBalgovindMatrix
  This class defines a covariance matrix in isotropic Balgovind form.
- class BasePerturbationManager
  This class generates and applies perturbations.
- class EnsembleKalmanFilter
  This class implements the ensemble Kalman filter.
- class ExtendedKalmanFilter
  This class implements the extended Kalman filter.
- class ExtendedMinimaxFilter
  This class implements the extended Kalman filter.
- class ForwardDriver
  This class simply performs a forward simulation.
- class FourDimensionalVariational
  This class implements 4D-Var.
- class FrontPositionObserver
  This class simply performs a forward simulation.
- class HamiltonJacobiBellman
  This class is a solver for Hamilton-Jacobi-Bellman equation.
- class MonteCarlo
  This class performs allows to perform Monte Carlo simulations.
- class NewranPerturbationManager
  This class generates random samples using Newran.
- class Nudging
- class ObservationGenerator
  This class simply performs a forward simulation.
- class OptimalInterpolation
This class performs optimal interpolation.

- class RandomPerturbationManager
  This class generates random samples using the random library from C++.

- class ReducedMinimax
  This class implements a reduced minimax filter.

- class ReducedOrderExtendedKalmanFilter
  This class implements a reduced order extended Kalman filter.

- class ReducedOrderUnscentedKalmanFilter
  This class implements a reduced order unscented Kalman filter.

- class TR1PerturbationManager
  This class generates random samples using C++ TR1 library.

- class TrajectoryManager
  This class manages model trajectory.

- class TRNGPerturbationManager
  This class generates random samples using TRNG.

- class UnscentedKalmanFilter
  This class implements the unscented Kalman filter.

- class CheckingModel
  This class is a model template.

- class ClampedBar
  This class is a clamped-bar model.

- class Lorenz
  This class is a Lorenz model.

- class ModelTemplate
  This class is a model template.

- class PetscClampedBar
  This class is a clamped-bar model.

- class PythonModel
- class QuadraticModel
  This class is a quadratic model.

- class ShallowWater
  This class is a shallow-water model.

- class GridToNetworkObservationManager
  Observation operator that maps from a grid to a network.

- class LevelSetObservationManager
  This class is a template of observation manager.

- class LinearObservationManager
  Linear observation operator.

- class ObservationAggregator
  Observation manager which computes an aggregated observation vector.

- class ObservationManagerTemplate
  This class is a template of observation manager.

- class PetscLinearObservationManager
  Linear observation operator.

- class PythonObservationManager
  This class is an interface for Python observation managers.

- class BaseForecaster
- class DiscountedRidgeRegression
- class Error
  This class serves for exceptions raised when an error is found.

- class ErrorConfiguration
This class serves for exceptions raised when an error is found in a configuration.

- class `ErrorIO`
  This class serves for exceptions raised when an input/output error occurs.

- class `ErrorProcessing`
  This class serves for exceptions raised when an error occurs during some data processing.

- class `ErrorUndefined`
  This class serves for exceptions raised when an undefined function or method is called.

- class `ErrorArgument`
  This class serves for exceptions raised when a function or a method is called with an erroneous argument.

- class `ErrorPythonUndefined`
  This class serves for exceptions raised when an undefined function or method in a Python module is called.

- class `Logger`
  Logging class.

- class `MessageHandler`
  This class enables objects to communicate using messages.

- class `OutputSaver`
  This class provides convenient methods to save variables on disk.

- class `Variable`
  This class enables to define a variable for the output saver.

- class `VerdandiBase`
  Base class for Verdandi objects.

- class `VerdandiOps`
  This class extends the Ops::Ops class with Verdandi-related features.

### Functions

- template\<\<\class Model, class ObservationManager, class Innovation, class State >\>
  void `ComputeBLUE_vector` (Model &model, ObservationManager &observation_manager, const Innovation &innovation, State &state)
  Computes BLUE.

- template\<\<\class Model, class ObservationManager, class Innovation, class State >\>
  void `ComputeBLUE_vector` (Model &model, ObservationManager &observation_manager, const Innovation &innovation, State &state, State &variance)
  Computes BLUE and the diagonal of its variance.

- template\<\<\class StateErrorVariance, class ObservationOperator, class Observation, class ObservationErrorVariance, class State >\>
  void `ComputeBLUE_matrix` (StateErrorVariance &B, const ObservationOperator &H, const Observation &y, const ObservationErrorVariance &R, State &x, bool is_y_innovation, bool compute_variance)
  Computes BLUE using operations on matrices.

- template\<\<\class StateErrorVariance, class ObservationOperator, class MatrixStateObservation, class Observation, class ObservationErrorVariance, class State >\>
  void `ComputeBLUE_matrix` (StateErrorVariance &B, const ObservationOperator &H, const MatrixStateObservation &cm, const Observation &y, const ObservationErrorVariance &R, State &x, bool is_y_innovation, bool compute_variance)
  Computes BLUE using operations on matrices.

- template\<\<\class StateErrorVariance, class ObservationOperator, class ObservationVector, class ObservationErrorVariance, class StateVector >\>
  void `ComputeCanonicalSigmaPoint` (const StateVector &x, const StateErrorVariance &B, const ObservationOperator &H, const ObservationVector &y, const ObservationErrorVariance &R)
  Computes \((y - Hx)^T (R + HBH^T)^{-1} (y - Hx)\).

- template\<\<\class SigmaPoint >\>
  void `ComputeCanonicalSigmaPoint` (int p, Vector< SigmaPoint, Collection > &sigma_point, SigmaPoint &alpha, bool &alpha_constant)
Computes 'canonical' sigma-points.

```cpp
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowMajor, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```cpp
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```cpp
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T,VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.

```
void ComputeCanonicalSigmaPoint (int p, Matrix<T, General, RowSparse, Allocator<T>> &sigma_point, Vector<T, VectFull, Allocator<T>> &alpha, bool &alpha_constant)
```

Computes 'canonical' sigma-points.
- \texttt{template<class T, class Allocator0, class Allocator1>}
  \texttt{T \text{Mean} (const Vector2<T, Allocator0, Allocator1> &V)}
- \texttt{template<class T, class TV, class Allocator0, class Allocator1>}
  \texttt{void \text{RemoveData} (T value, Vector2<TV, Allocator0V, Allocator1V> &V)}
- \texttt{template<class T, class TV, class Allocator0V, class Allocator1V, class TW, class Allocator0W, class Allocator1W>}
  \texttt{void \text{RemoveData} (T value, Vector2<TV, Allocator0V, Allocator1V> &V, Vector2<TW, Allocator0W, Allocator1W> &W)}
- \texttt{template<class T, class TV, class Allocator0V, class Allocator1V, class TW, class Allocator0W, class Allocator1W, class TZ, class Allocator0Z, class Allocator1Z>}
  \texttt{void \text{RemoveData} (T value, Vector2<TV, Allocator0V, Allocator1V> &V, Vector2<TW, Allocator0W, Allocator1W, Allocator0Z, Allocator1Z> &Z)}
- \texttt{template<class T, class TV, class Allocator0V, class Allocator1V, class TW, class Allocator0W, class Allocator1W>}
  \texttt{void \text{RemoveData} (T value_min, T value_max, Vector2<T, Allocator0V, Allocator1V> &V, Vector2<TW, Allocator0W, Allocator1W> &W)}
- \texttt{template<class T, class TV, class Allocator0V, class Allocator1V, class TW, class Allocator0W, class Allocator1W, class TZ, class Allocator0Z, class Allocator1Z>}
  \texttt{void \text{RemoveData} (T value_min, T value_max, Vector2<T, Allocator0V, Allocator1V> &V, Vector2<TW, Allocator0W, Allocator1W, Allocator0Z, Allocator1Z> &Z)}
- \texttt{template<class Allocator0l, class Allocator1l, class Td, class Allocator0d, class Allocator1d>}
  \texttt{void \text{RemoveEmptyLocation} (Vector2<int, Allocator0l, Allocator1l> &location_index, Vector2<T, Allocator0d, Allocator1d> &data, double ratio=numeric_limits<double>::epsilon())}
- \texttt{template<class Allocator0l, class Allocator1l, class Td, class Allocator0d, class Allocator1d>}
  \texttt{void \text{RemoveEmptyLocation} (Vector2<int, Allocator0l, Allocator1l> &location_index, Vector2<T, Allocator0d, Allocator1d> &data, double ratio=numeric_limits<double>::epsilon())}
- \texttt{template<class Allocator0l, class Allocator1l, class Td, class Allocator0d, class Allocator1d>}
  \texttt{void \text{RemoveLocation} (const Vector2<bool> &keep, Vector2<int, Allocator0l, Allocator1l> &location_index, Vector2<Td, Allocator0d, Allocator1d> &data)}
- \texttt{template<class Allocator0r, class Allocator1r, class Allocator0i, class Allocator1i>}
  \texttt{void \text{SelectLocation} (const Vector2<int, Allocator0r, Allocator1r> &index_ref, Vector2<int, Allocator0i, Allocator1i> &index)}
- \texttt{template<class Allocator0r, class Allocator1r, class Allocator0i, class Allocator1i, class Td, class Allocator0d, class Allocator1d>}
  \texttt{void \text{SelectLocation} (const Vector2<int, Allocator0r, Allocator1r> &index_ref, Vector2<int, Allocator0i, Allocator1i> &index, Vector2<Td, Allocator0d, Allocator1d> &data)}
- \texttt{template<class TV, class Allocator0V, class Allocator1V, class T, class Allocator>}
  \texttt{void \text{Collect} (const Vector2<TV, Allocator0V, Allocator1V> &V, Vector2<T, VectFull, Allocator> &data)}
- \texttt{template<class TV, class Allocator0V, class Allocator1V, class T, class Allocator>}
  \texttt{void \text{CollectDate} (const Vector2<TV, Allocator0V, Allocator1V> &V, int beg, int end, Vector2<T, VectFull, Allocator> &data)}
- \texttt{template<class Allocator0l, class Allocator1l, class Td, class Allocator0d, class Allocator1d>}
  \texttt{void \text{CollectLocation} (const Vector2<int, Allocator0l, Allocator1l> &location_index, const Vector2<Td, Allocator0d, Allocator1d> &data, int location, Vector<int, Allocators> &step, Vector<T, Allocator0> &output)}
- \texttt{template<class Allocator0l, class Allocator1l, class Td, class Allocator0d, class Allocator1d, class Allocators, class To, class Allocator>}
  \texttt{void \text{CollectLocation} (const Vector2<int, Allocator0l, Allocator1l> &location_index, const Vector2<Td, Allocator0d, Allocator1d> &data, int location, Vector<int, Allocators> &step, Vector<T, Allocator0> &output)}
- \texttt{template<class TV, class Allocator0V, class Allocator1V, class TW, class Allocator0W, class Allocator1W>}
  \texttt{void \text{CheckShape} (const Vector2<TV, Allocator0V, Allocator1V> &V, const Vector2<TW, Allocator0W, Allocator1W> &W, string function="")}
- \texttt{template<class T, class TV, class Allocator0, class Allocator1, class Allocator2>}
  \texttt{void \text{RemoveData} (T value, Vector3<T, TV, Allocator0, Allocator1, Allocator2> &V)}
- \texttt{bool \text{Lock} (string const &filename)}
  \texttt{Creates a lock file.}
- \texttt{bool \text{Unlock} (string const &filename)}
- bool **Unlock**(std::string const &filename)
  Removes a lock file.
- void **get_position**(int index, const Vector<int>& shape, Vector<int>& position)
  Returns the position in a multidimensional grid that is associated with a global index.
- int **get_position**(const Vector<int>& shape, const Vector<int>& position)
  Returns the global index in a multidimensional grid that is associated with a local position.
- void **convert**(const string &s, string &out)
  Sets a string.
- bool **is_num**(const string &str)
  Checks whether a string is a number.
- bool **is_integer**(const string &str)
  Checks whether a string is an integer.
- bool **is_unsigned_integer**(const string &str)
  Checks whether a string is an unsigned integer.
- string **trim**(string str, string delimiters)
  Trims off a string.
- vector<string> **split**(string str, string delimiters)
  Splits a string.
- string **find_replace**(string str, string old_str, string new_str)
  Finds and replace a substring.
- string **upper_case**(string str)
  Converts a string to upper-case string.
- template<class T, class TM>
  T **interpolate**(T x_min, T Delta_x, T y_min, T Delta_y, const Matrix<TM>& input, T x, T y)
- template<class T>
  void **get_coordinate**(int index, const Vector<T>& minimum, const Vector<T>& step, const Vector<int>& shape, Vector<T>& coordinate)
- template<class T>
  void **split**(string str, vector<T>& vect, string delimiters="\n\t")
- template<class T>
  bool **is_equal**(T x, T y, T epsilon=1.e-6)
- template<class T>
  bool **is_multiple**(T x, T d, T epsilon=1.e-6)
- template<class T, class Allocator>
  void **build_diagonal_sparse_matrix**(int size, T diagonal_value, Matrix<T, General, RowSparse>& matrix)
- template<class T0, class Allocator0, class T1, class Allocator1>
  void **Copy**(const Matrix<T0, General, RowMajor, Allocator0>& A, Matrix<T1, General, RowSymPacked, Allocator1>& B)
- template<class T0, class Allocator0, class T1, class Allocator1>
  void **Copy**(const Matrix<T0, General, RowSparse, Allocator0>& A, Matrix<T1, General, RowMajor, Allocator1>& A_dense)
- template<class T0, class T1, class Allocator1, class T2, class Allocator2>
  void **Add**(const T0 alpha, const Vector<T1, Collection, Allocator1>& X, Vector<T2, VectFull, Allocator2>& Y)
- template<class T0, class Allocator0, class T1, class Allocator1>
  void **SetCol**(const Vector<T1, VectFull, Allocator1>& X, int i, Matrix<T0, General, RowSparse, Allocator0>& M)
47.1.1 Detailed Description

The namespace of the data assimilation library Verdandi.

47.1.2 Function Documentation

47.1.2.1 template< class Vector3 , class T , class Vector2 > void Verdandi::Aggregate ( const Vector3 & ensemble, const Vector< T > & weight, Vector2 & aggregated )

Aggregates ensemble data.

The aggregation is carried out with a given weight vector. Every member is given a single weight, independent of time and location.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ensemble</td>
<td>the ensemble data indexed by member, time and location.</td>
</tr>
<tr>
<td>in</td>
<td>weight</td>
<td>aggregation weights, indexed by member.</td>
</tr>
<tr>
<td>out</td>
<td>aggregated</td>
<td>the aggregated output, indexed by time and location.</td>
</tr>
</tbody>
</table>

Definition at line 36 of file Aggregate.hxx.

47.1.2.2 template< class Vector3 , class T , class Vector2 > void Verdandi::Aggregate ( const Vector3 & ensemble, const Vector< T > & weight, int begin, int end, Vector2 & aggregated )

Aggregates ensemble data.
The aggregation is carried out with a given weight vector. Every member is given a single weight, independent of time and location. The output aggregated values are kept only for the time indexes begin to end - 1.

### Parameters

| in | ensemble | the ensemble data indexed by member, time and location. |
| in | weight | aggregation weights, indexed by member. |
| in | begin | inclusive lower-bound for the time indexes. |
| in | end | exclusive upper-bound for the time indexes. |
| out | aggregated | the aggregated output, indexed by time and location, but only for the time period [beg, end]. |

Definition at line 89 of file Aggregate.hxx.

47.1.2.3 template class Vector3, class T, class Vector2 > void Verdandi::Aggregate ( const Vector3 & ensemble, const Matrix & weight, Vector2 & aggregated )

Aggregates ensemble data.

The aggregation is carried out with a given weight vector. Every member is given a single weight per time step.

### Parameters

| in | ensemble | the ensemble data indexed by member, time and location. |
| in | weight | aggregation weights, indexed by time and member. |
| out | aggregated | the aggregated output, indexed by time and location. |

Definition at line 145 of file Aggregate.hxx.

47.1.2.4 template class StateErrorVariance, class ObservationOperator, class ObservationVector, class ObservationErrorVariance, class StateVector > StateVector::value_type Verdandi::chi_2 ( const StateVector & x, const StateErrorVariance & B, const ObservationOperator & H, const ObservationVector & y, const ObservationErrorVariance & R )

Computes \( (y - Hx)^T (R + HBH^T)^{-1} (y - Hx) \).

This method can help to check the consistency between an innovation and its statistics.

### Parameters

| in | x | the state vector. |
| in | B | error variance associated with x. |
| in | H | observation operator. |
| in | y | the vector of observations. |
| in | R | error variance associated with y. |

Returns

\( (y - Hx)^T (R + HBH^T)^{-1} (y - Hx) \)

Definition at line 47 of file chi_2.hxx.

47.1.2.5 template class StateErrorVariance, class ObservationOperator, class Observation, class ObservationErrorVariance, class State > void Verdandi::ComputeBLUE_matrix ( StateErrorVariance & B, const ObservationOperator & H, const Observation & y, const ObservationErrorVariance & R, State & x, bool is_y_innovation, bool compute_variance )

Computes BLUE using operations on matrices.
This method is mainly intended for cases where the covariance matrices are sparse matrices. Otherwise, the manipulation of the matrices may lead to unreasonable memory requirements and to high computational costs.

**Parameters**

- **B**: error variance associated with state.
- **H**: observation operator.
- **y**: the vector of observations or innovations.
- **R**: error variance associated with observation.
- **x**: on entry, the background vector; on exit, the analysis.
- **is_y_innovation**: Boolean to indicate if the parameter \( y \) is a vector of observations or innovations.
- **compute_variance**: Boolean to indicate if the covariance matrix has to be updated.

Definition at line 332 of file BLUE.cxx.

### 47.1.2.6 template< class StateErrorVariance, class ObservationOperator, class MatrixStateObservation, class Observation, class ObservationErrorVariance, class State >

\[
\text{void Verdandi::ComputeBLUE\_matrix ( StateErrorVariance & B, \text{const ObservationOperator & H, \text{const MatrixStateObservation & cm, \text{const Observation & y, \text{const ObservationErrorVariance & R, \text{State & x, bool is\_y\_innovation, bool compute\_variance } })}}
\]

Computes BLUE using operations on matrices.

This method is mainly intended for cases where the covariance matrices are sparse matrices. Otherwise, the manipulation of the matrices may lead to unreasonable memory requirements and to high computational costs.

**Parameters**

- **B**: error variance associated with state.
- **H**: observation operator.
- **cm**: this parameter is only used to determine the type of an intermediate matrix in the computations. cm is not modified nor read. Its type CrossedMatrix will be the type of the intermediate matrix \( BH' \), whose size is \( Nx \) times \( Ny \), if \( Nx \) is the length of \( x \) and \( Ny \) is the length of \( y \).
- **y**: the vector of observations or innovations.
- **R**: error variance associated with observation.
- **x**: on entry, the background vector; on exit, the analysis.
- **is_y_innovation**: Boolean to indicate if the parameter \( y \) is a vector of observations or innovations.
- **compute_variance**: Boolean to indicate if the covariance matrix has to be updated.

Definition at line 369 of file BLUE.cxx.

### 47.1.2.7 template< class Model, class ObservationManager, class Innovation, class State >

\[
\text{void Verdandi::ComputeBLUE\_vector ( Model & model, ObservationManager & observation\_manager, const Innovation & innovation, State & state )}
\]

Computes BLUE.

It computes the BLUE (best linear unbiased estimator).

**Parameters**

- **model**: the model.
- **observation\_manager**: the observation manager.
- **innovation**: the innovation vector.
- **x**: on entry, the background vector; on exit, the analysis.
47.1.2.8 template < class Model, class ObservationManager, class Innovation, class State > void 
   Verdandi::ComputeBLUE_vector ( Model & model, ObservationManager & observation_manager, 
   const Innovation & innovation, State & state, State & variance ) 

Computes BLUE and the diagonal of its variance.

It computes the BLUE (best linear unbiased estimator) and the diagonal of its variance.

Parameters

| in   | model | the model. |
|      |       |            |
| in   | observation_manager | the observation manager. |
| in, out | innovation | the innovation vector. |
| out  | variance | on exit, the diagonal elements of the analysis variance, i.e., the variances of the components of x. |

Definition at line 179 of file BLUE.cxx.

47.1.2.9 template < class SigmaPoint > void Verdandi::ComputeCanonicalSigmaPoint ( int p, 
   Vector < SigmaPoint, Collection > & sigma_point, SigmaPoint & alpha, bool & alpha_constant )

Computes 'canonical' sigma-points.

Parameters

| in   | p | dimension of the model state. |
| out  | sigma_point | 'canonical' sigma-points. |
| out  | alpha | coefficient vector associated with sigma-points. |
| out  | alpha_constant | boolean to indicate if the coefficients alpha are constants. |

Definition at line 44 of file SigmaPoint.cxx.

47.1.2.10 template < class T, template < class U > class Allocator > void Verdandi::ComputeCanonicalSigmaPoint ( int p, 
   Matrix < T, General, RowMajor, Allocator < T > > & sigma_point, Vector < T, VectFull, Allocator < T > > & alpha, 
   bool & alpha_constant )

Computes 'canonical' sigma-points.

Parameters

| in   | p | dimension of the model state. |
| out  | sigma_point | 'canonical' sigma-points. |
| out  | alpha | coefficient vector associated with sigma-points. |
| out  | alpha_constant | boolean to indicate if the coefficients alpha are constants. |

Definition at line 90 of file SigmaPoint.cxx.

47.1.2.11 template < class T, template < class U > class Allocator > void Verdandi::ComputeCanonicalSigmaPoint ( int p, 
   Matrix < T, General, RowSparse, Allocator < T > > & sigma_point, Vector < T, VectFull, Allocator < T > > & alpha, 
   bool & alpha_constant )

Computes 'canonical' sigma-points.
### 47.1.2.12 template< class SigmaPoint > void Verdandi::ComputeSimplexSigmaPoint ( int p, Vector< SigmaPoint, Collection > & sigma_point, SigmaPoint & alpha, bool & alpha_constant )

Computes 'simplex' sigma-points.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
</tr>
<tr>
<td><strong>p</strong></td>
</tr>
<tr>
<td><strong>out</strong></td>
</tr>
<tr>
<td><strong>sigma_point</strong></td>
</tr>
<tr>
<td><strong>out</strong></td>
</tr>
<tr>
<td><strong>alpha</strong></td>
</tr>
<tr>
<td><strong>out</strong></td>
</tr>
<tr>
<td><strong>alpha_constant</strong></td>
</tr>
</tbody>
</table>

Definition at line 122 of file SigmaPoint.cxx.

### 47.1.2.13 template< class T , template< class U > class Allocator > void Verdandi::ComputeSimplexSigmaPoint ( int p, Matrix< T, General, RowMajor, Allocator< T > > & sigma_point, Vector< T, VectFull, Allocator< T > > & alpha, bool & alpha_constant )

Computes 'simplex' sigma-points.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
</tr>
<tr>
<td><strong>p</strong></td>
</tr>
<tr>
<td><strong>out</strong></td>
</tr>
<tr>
<td><strong>sigma_point</strong></td>
</tr>
<tr>
<td><strong>out</strong></td>
</tr>
<tr>
<td><strong>alpha</strong></td>
</tr>
</tbody>
</table>

Definition at line 234 of file SigmaPoint.cxx.

### 47.1.2.14 template< class T , template< class U > class Allocator > void Verdandi::ComputeSimplexSigmaPoint ( int p, Matrix< T, General, RowSparse, Allocator< T > > & sigma_point, Vector< T, VectFull, Allocator< T > > & alpha, bool & alpha_constant )

Computes 'simplex' sigma-points.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
</tr>
<tr>
<td><strong>p</strong></td>
</tr>
<tr>
<td><strong>out</strong></td>
</tr>
<tr>
<td><strong>sigma_point</strong></td>
</tr>
<tr>
<td><strong>out</strong></td>
</tr>
<tr>
<td><strong>alpha</strong></td>
</tr>
</tbody>
</table>

Definition at line 284 of file SigmaPoint.cxx.

### 47.1.2.15 template< class SigmaPoint > void Verdandi::ComputeStarSigmaPoint ( int p, Vector< SigmaPoint, Collection > & sigma_point, SigmaPoint & alpha, bool & alpha_constant )

Computes 'star' sigma-points.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
</tr>
<tr>
<td><strong>p</strong></td>
</tr>
<tr>
<td><strong>out</strong></td>
</tr>
<tr>
<td><strong>sigma_point</strong></td>
</tr>
<tr>
<td><strong>out</strong></td>
</tr>
<tr>
<td><strong>alpha</strong></td>
</tr>
</tbody>
</table>

Definition at line 314 of file SigmaPoint.cxx.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>p</th>
<th>dimension of the model state.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>sigma_point</td>
<td>'star' sigma-points.</td>
</tr>
<tr>
<td>out</td>
<td>alpha</td>
<td>coefficient vector associated with sigma-points.</td>
</tr>
<tr>
<td>out</td>
<td>alpha_constant</td>
<td>boolean to indicate if the coefficients alpha are constants.</td>
</tr>
</tbody>
</table>

Definition at line 146 of file SigmaPoint.cxx.

47.1.2.16 template<class T , template<class U> class Allocator> void Verdandi::ComputeStarSigmaPoint( int p, Matrix< T, General, RowMajor, Allocator< T > > & sigma_point, Vector< T, VectFull, Allocator< T > > & alpha, bool & alpha_constant )
Computes 'star' sigma-points.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>p</th>
<th>dimension of the model state.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>sigma_point</td>
<td>'star' sigma-points.</td>
</tr>
<tr>
<td>out</td>
<td>alpha</td>
<td>coefficient vector associated with sigma-points.</td>
</tr>
<tr>
<td>out</td>
<td>alpha_constant</td>
<td>boolean to indicate if the coefficients alpha are constants.</td>
</tr>
</tbody>
</table>

Definition at line 180 of file SigmaPoint.cxx.

47.1.2.17 template<class T , template<class U> class Allocator> void Verdandi::ComputeStarSigmaPoint( int p, Matrix< T, General, RowSparse, Allocator< T > > & sigma_point, Vector< T, VectFull, Allocator< T > > & alpha, bool & alpha_constant )
Computes 'star' sigma-points.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>p</th>
<th>dimension of the model state.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>sigma_point</td>
<td>'star' sigma-points.</td>
</tr>
<tr>
<td>out</td>
<td>alpha</td>
<td>coefficient vector associated with sigma-points.</td>
</tr>
<tr>
<td>out</td>
<td>alpha_constant</td>
<td>boolean to indicate if the coefficients alpha are constants.</td>
</tr>
</tbody>
</table>

Definition at line 212 of file SigmaPoint.cxx.

47.1.2.18 void Verdandi::convert ( const string & s, string & out )
Sets a string.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>s</th>
<th>input string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>out</td>
<td>output string, equal to s on exit.</td>
</tr>
</tbody>
</table>

Definition at line 117 of file UsefulFunction.cxx.

47.1.2.19 string Verdandi::find_replace ( string str, string old_str, string new_str )
Finds and replace a substring.
**Parameters**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>str</code></td>
<td>base string.</td>
</tr>
<tr>
<td><code>old_str</code></td>
<td>substring to be replaced.</td>
</tr>
<tr>
<td><code>new_str</code></td>
<td>substring to be put in place of 'old_str'.</td>
</tr>
</tbody>
</table>

**Returns**

'str' where 'old_str' was replaced by 'new_str'.

Definition at line 247 of file UsefulFunction.cxx.

### 47.1.2.20 template <class Vector2, class T> void Verdandi::Flatten ( const Vector2 & Vin, int begin, int end, Vector<T> & Vout )

Returns in a vector all values from a range of inner vectors of `Vin`.

The output vector `Vout` contains all inner vectors of `Vin`, in the index range `[beg, end]`, concatenated in the same order as they appear in `Vin`.

**Parameters**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>out</code></td>
<td><code>Vin</code> Vector2 instance from which the data is extracted.</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>begin</code> inclusive lower-bound for the indexes.</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>end</code> exclusive upper-bound for the indexes.</td>
</tr>
<tr>
<td><code>out</code></td>
<td><code>Vout</code> the values contained in the inner vectors <code>[beg, end]</code>.</td>
</tr>
</tbody>
</table>

Definition at line 206 of file Aggregate.cxx.

### 47.1.2.21 template <class Vector3, class T> void Verdandi::Flatten ( const Vector3 & Vin, int begin, int end, Matrix<T> & Mout )

Returns in a matrix all values from a range of vectors of `Vin`.

Every row in the output matrix `Mout` contains all inner vectors of the i-th vector of vectors of `Vin`, in the index range `[beg, end]`, concatenated in the same order as they appear in the inner i-th vector of vectors of `Vin`. If `Vin` is a 3D array with shape (N0, N1, N2), the output matrix `Mout` is of shape (N0, (end - begin) ? N2).

**Parameters**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>in</code></td>
<td><code>Vin</code> Vector3 instance from which the data is extracted. The total number of elements in every selected vector of vectors should be the same, since it is the number of columns in the output matrix.</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>begin</code> inclusive lower-bound for the indexes.</td>
</tr>
<tr>
<td><code>in</code></td>
<td><code>end</code> exclusive upper-bound for the indexes.</td>
</tr>
<tr>
<td><code>out</code></td>
<td><code>Mout</code> the values contained in the inner vectors <code>[beg, end]</code>.</td>
</tr>
</tbody>
</table>

Definition at line 228 of file Aggregate.cxx.

### 47.1.2.22 void Verdandi::get_position ( int index, const Vector<int>& shape, Vector<int>& position )

Returns the position in a multidimensional grid that is associated with a global index.

A global index gives the position in a grid with a single integer. For example, in 2D, the global index of the grid point `(i, j)` is `i × N + j` if there are `N` points along the second dimension. This function would return `(i, j)` in `position` from `(index =) i × N + j`, with `shape` set to `(M, N)`.
Parameters

<table>
<thead>
<tr>
<th>index</th>
<th>the global index.</th>
</tr>
</thead>
<tbody>
<tr>
<td>shape</td>
<td>dimensions of the grid.</td>
</tr>
<tr>
<td>position</td>
<td>position in the grid.</td>
</tr>
</tbody>
</table>

Definition at line 46 of file UsefulFunction.cxx.

47.1.2.23 int Verdandi::get_position ( const Vector<int> & shape, const Vector<int> & position )

Returns the global index in a multidimensional grid that is associated with a local position.

A global index gives the position in a grid with a single integer. For example, in 2D, the global index of the grid point \((i, j)\) is \(i \times N + j\) if there are \(N\) points along the second dimension. This function returns \(i \times N + j\) from \(position\) set to \((i, j)\), if \(shape\) is \((M, N)\).

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>shape</th>
<th>dimensions of the grid.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>position</td>
<td>position in the grid.</td>
</tr>
</tbody>
</table>

Returns

index The global index.

Definition at line 99 of file UsefulFunction.cxx.

47.1.2.24 bool Verdandi::is_integer ( const string & str )

Checks whether a string is an integer.

Parameters

| in | str | string to be checked. |

Returns

True if \(str\) is an integer, false otherwise.

Definition at line 168 of file UsefulFunction.cxx.

47.1.2.25 bool Verdandi::is_num ( const string & str )

Checks whether a string is a number.

Parameters

| in | str | string to be checked. |

Returns

True if \(str\) is a number, false otherwise.

Definition at line 128 of file UsefulFunction.cxx.
47.1.2.26  bool Verdandi::is_unsigned_integer ( const string & str )

Checks whether a string is an unsigned integer.

Parameters

| in  | str | string to be checked. |

Returns

True if str is an unsigned integer, false otherwise.

Definition at line 191 of file UsefulFunction.cxx.

47.1.2.27  bool Verdandi::Lock ( string const & filename )

Creates a lock file.

If the lock file already exists, this function waits first for this file to be removed. If the file is still there after one million seconds, the lock fails and this function returns 'false'.

Parameters

| in  | filename | name of the lock file. |

Returns

True if the lock file was successfully created, false otherwise.

Definition at line 52 of file LockFile.cxx.

47.1.2.28  vector<string> Verdandi::split ( string str, string delimiters )

Splits a string.

The string is split according to delimiters.

Parameters

| in  | str | string to be split. |
| in  | delimiters | (optional) delimiters. Default: "\n\t". |

Returns

A vector containing elements of the string.

Definition at line 232 of file UsefulFunction.cxx.

47.1.2.29  string Verdandi::trim ( string str, string delimiters )

Trims off a string.

Removes delimiters at each edge of the string.

Parameters

| in  | str | string to be trimmed off. |
| in  | delimiters | characters to be removed. |
Returns

str trimmed off.

Definition at line 213 of file UsefulFunction.cxx.

47.1.2.30 bool Verdandi::Unlock ( string const & filename )

Removes a lock file.

Parameters

| in   | filename | name of the lock file. |

Definition at line 84 of file LockFile.cxx.

47.1.2.31 string Verdandi::upper_case ( string str )

Converts a string to upper-case string.

Parameters

|       | str     | string to be converted. |

Returns

str in upper case.

Definition at line 266 of file UsefulFunction.cxx.
48.1 Verdandi::BalgovindMatrix Class Reference

This class defines a covariance matrix in Balgovind form.

```cpp
#include <BalgovindMatrix.hxx>
```

Public Member Functions

- ```cpp
   Verdandi::BalgovindMatrix (T x_min, T delta_x, int Nx, T y_min, T delta_y, int Ny, T length_x, T length_y, T variance)
   Constructor for 2D cases.
   ```

- ```cpp
   Verdandi::BalgovindMatrix (T x_min, T delta_x, int Nx, T y_min, T delta_y, int Ny, T z_min, T delta_z, int Nz, T length_x, T length_y, T length_z, T variance)
   Constructor for 3D cases.
   ```

- ```cpp
   int GetM () const
   Returns the number of rows.
   ```

- ```cpp
   int GetN () const
   Returns the number of columns.
   ```

- ```cpp
   T operator() (int i, int j) const
   Access to an entry of the matrix.
   ```

- ```cpp
   void GetRow (int i, Vector<T> &row) const
   Access to a row.
   ```

- ```cpp
   void GetCol (int i, Vector<T> &column) const
   Access to a column.
   ```

Protected Attributes

- ```cpp
   Vector<T> min_
   Coordinates of the domain first cell center.
   ```

- ```cpp
   Vector<T> step_
   Space steps.
   ```

- ```cpp
   Vector<int> N_
   Number of points along each dimension.
   ```

- ```cpp
   Vector<T> length_scale_
   Length scales.
   ```

- ```cpp
   T variance_
   Variance.
   ```

- ```cpp
   int dimension_
   Matrix dimension.
48.1.1 Detailed Description

This class defines a covariance matrix in Balgovind form.

In a covariance matrix in Balgovind form, the covariance between two points depends on the distances (in each direction) between the points. For example, in the 2D case, if the entry \((i, j)\) of the matrix is the covariance between the values at \((x_i, y_i)\) and \((x_j, y_j)\), its value will be:

\[
B_{i,j} = v \left( 1 + \frac{|x_j - x_i|}{L_x} \right) \exp \left( -\frac{|x_j - x_i|}{L_x} \right) \left( 1 + \frac{|y_j - y_i|}{L_y} \right) \exp \left( -\frac{|y_j - y_i|}{L_y} \right)
\]

where \(v\) is a variance, and \(L_x\) and \(L_y\) are decorrelation lengths.

48.1.2 Constructor & Destructor Documentation

48.1.2.1 Verdandi::BalgovindMatrix::BalgovindMatrix ( T x_min, T delta_x, int Nx, T y_min, T delta_y, int Ny, T length_x, T length_y, T variance )

Constructor for 2D cases.
This constructors builds a Balgovind matrix for 2D regular grids.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_min</td>
<td>abscissa of the center of the lower-left grid cell.</td>
</tr>
<tr>
<td>delta_x</td>
<td>step along x.</td>
</tr>
<tr>
<td>Nx</td>
<td>number of cells along x.</td>
</tr>
<tr>
<td>y_min</td>
<td>ordinate of the center of the lower-left grid cell.</td>
</tr>
<tr>
<td>delta_y</td>
<td>step along x.</td>
</tr>
<tr>
<td>Ny</td>
<td>number of cells along y.</td>
</tr>
<tr>
<td>length_x</td>
<td>decorrelation length along x.</td>
</tr>
<tr>
<td>length_y</td>
<td>decorrelation length along y.</td>
</tr>
<tr>
<td>variance</td>
<td>variance.</td>
</tr>
</tbody>
</table>

Definition at line 51 of file BalgovindMatrix.cxx.

48.1.2.2 Verdandi::BalgovindMatrix::BalgovindMatrix ( T x_min, T delta_x, int Nx, T y_min, T delta_y, int Ny, T z_min, T delta_z, int Nz, T length_x, T length_y, T length_z, T variance )

Constructor for 3D cases.
This constructors builds a Balgovind matrix for 3D regular grids.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_min</td>
<td>abscissa of the center of the lower-left grid cell.</td>
</tr>
<tr>
<td>delta_x</td>
<td>step along x.</td>
</tr>
<tr>
<td>Nx</td>
<td>number of cells along x.</td>
</tr>
<tr>
<td>y_min</td>
<td>ordinate of the center of the lower-left grid cell.</td>
</tr>
<tr>
<td>delta_y</td>
<td>step along x.</td>
</tr>
<tr>
<td>Ny</td>
<td>number of cells along y.</td>
</tr>
<tr>
<td>z_min</td>
<td>applicate of the center of the bottom grid cells.</td>
</tr>
<tr>
<td>delta_z</td>
<td>step along z.</td>
</tr>
<tr>
<td>Nz</td>
<td>number of cells along z.</td>
</tr>
<tr>
<td>length_x</td>
<td>decorrelation length along x.</td>
</tr>
<tr>
<td>length_y</td>
<td>decorrelation length along y.</td>
</tr>
<tr>
<td>length_z</td>
<td>decorration length along z.</td>
</tr>
<tr>
<td>variance</td>
<td>variance.</td>
</tr>
</tbody>
</table>
48.1 Verdandi::BalgovindMatrix Class Reference

Definition at line 88 of file BalgovindMatrix.cxx.

48.1.3 Member Function Documentation

48.1.3.1 void Verdandi::BalgovindMatrix::GetCol ( int j, Vector<T> & column ) const

Access to a column.

<table>
<thead>
<tr>
<th>in</th>
<th>j</th>
<th>column index.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>column</td>
<td>the j-th row.</td>
</tr>
</tbody>
</table>

Definition at line 183 of file BalgovindMatrix.cxx.

48.1.3.2 int Verdandi::BalgovindMatrix::GetM ( ) const [inline]

Returns the number of rows.

Returns

   The number of rows.

Definition at line 123 of file BalgovindMatrix.cxx.

48.1.3.3 int Verdandi::BalgovindMatrix::GetN ( ) const [inline]

Returns the number of columns.

Returns

   The number of columns.

Definition at line 134 of file BalgovindMatrix.cxx.

48.1.3.4 void Verdandi::BalgovindMatrix::GetRow ( int i, Vector<T> & row ) const

Access to a row.

<table>
<thead>
<tr>
<th>in</th>
<th>i</th>
<th>row index.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>row</td>
<td>the i-th row.</td>
</tr>
</tbody>
</table>

Definition at line 169 of file BalgovindMatrix.cxx.

48.1.3.5 T Verdandi::BalgovindMatrix::operator () ( int i, int j ) const

Access to an entry of the matrix.

<table>
<thead>
<tr>
<th>in</th>
<th>i</th>
<th>row index.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>j</td>
<td>column index.</td>
</tr>
</tbody>
</table>
Returns

The value of the entry \((i, j)\) of the matrix.

Definition at line 147 of file BalgovindMatrix.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/error/BalgovindMatrix.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/error/BalgovindMatrix.cxx

### 48.2 Verdandi::BaseForecaster Class Reference

#### Public Member Functions

- **BaseForecaster ()**
  
  Main constructor.

- **virtual ~BaseForecaster ()**
  
  Destructor.

- **int GetNspinup () const**
  
  Returns the width of the spin-up period.

- **void SetNspinup (int Nspinup)**
  
  Sets the width of the spin-up period.

- **int GetNparameter () const**
  
  Return the number of parameters.

- **const Vector</T> & GetParameter () const**
  
  Returns the vector of parameters.

- **virtual void SetParameter (const Vector</T> &parameter)**
  
  Sets the vector of parameters.

- **bool IsConvex () const**
  
  Is the method carrying out convex aggregation?

  
  Aggregates the ensemble data.

- **virtual void AggregateThreshold (const Vector3</T> &ensemble, const Vector2</T> &observation, T threshold_value, Vector2</T> &aggregated_simulation)**
  
  Aggregates the ensemble data.

  
  Aggregates in the spin-up period at a given time.

  
  Aggregates in the spin-up period at a given time.
virtual void ComputeWeight (int t, const Matrix< T > &simulation_data, const Vector< T > &observation_data, const Vector3< T > &ensemble, const Vector2< T > &observation, const Matrix< T > &weight, Vector< T > &weight_vector)

Computes aggregation weights at a given time.

virtual void ComputeWeightNonSequential (int t, const Vector3< T > &ensemble, const Vector2< T > &observation, Vector< T > &weight_vector)

Computes aggregation weights at a given time.

virtual void ComputeAggregatedValue (int t, const Vector3< T > &ensemble, const Vector< T > &weight_vector, Vector2< T > &aggregated_simulation)

Computes the aggregated value based on given weights.

virtual void Aggregate (int t, const Matrix< T > &simulation_data, const Vector< T > &observation_data, const Vector3< T > &ensemble, const Vector2< T > &observation, Matrix< T > &weight, Vector2< T > &aggregated_simulation)

Aggregates at a given time.

virtual void AggregateThreshold (int t, const Matrix< T > &simulation_data, const Vector< T > &observation_data, const Vector3< T > &ensemble, const Vector2< T > &observation, T &threshold_value, Matrix< T > &weight, Matrix< T > &base_weight, Vector2< T > &aggregated_simulation)

Aggregates at a given time.

void AggregateMean (int t, const Matrix< T > &simulation_data, const Vector3< T > &ensemble, Matrix< T > &weight, Vector2< T > &aggregated_simulation)

Computes the ensemble mean at a given time.

void AggregateZero (int t, const Vector3< T > &ensemble, Matrix< T > &weight, Vector2< T > &aggregated_simulation)

Aggregates with zero weights at a given time.

Protected Attributes

int Nspinup_

Number of step in the spin-up period.

Vector< T > parameter_

Parameters.

bool is_convex_

Is the aggregation procedure convex?

48.2.1 Constructor & Destructor Documentation

48.2.1.1 Verdandi::BaseForecaster::BaseForecaster ( )

Main constructor.

The width of the learning time window is set to 1.

Definition at line 41 of file BaseForecaster.cxx.

48.2.2 Member Function Documentation

48.2.2.1 void Verdandi::BaseForecaster::Aggregate ( const Vector3< T > &ensemble, const Vector2< T > &observation, Vector2< T > &aggregated ) [virtual]

Aggregates the ensemble data.

It carries out the sequential aggregation and returns the aggregated forecast.
Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>ensemble</strong></td>
<td>the ensemble data indexed by member, time and location.</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>observation</strong></td>
<td>the observational data, indexed by time and location.</td>
</tr>
<tr>
<td><strong>out</strong></td>
<td><strong>aggregated</strong></td>
<td>the aggregated output, indexed by time and location.</td>
</tr>
</tbody>
</table>

Definition at line 133 of file BaseForecaster.cxx.

48.2.2.2  void Verdandi::BaseForecaster::Aggregate ( const Vector3<T> & ensemble, const Vector2<T> & observation, Matrix<T> & weight, Vector2<T> & aggregated ) [virtual]

Aggregates the ensemble data.

It carries out the sequential aggregation and returns the aggregated forecast together with the aggregation weights.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>ensemble</strong></td>
<td>the ensemble data indexed by member, time and location.</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>observation</strong></td>
<td>the observational data, indexed by time and location.</td>
</tr>
<tr>
<td><strong>out</strong></td>
<td><strong>weight</strong></td>
<td>aggregation weights, indexed by member and time.</td>
</tr>
<tr>
<td><strong>out</strong></td>
<td><strong>aggregated</strong></td>
<td>the aggregated output, indexed by time and member.</td>
</tr>
</tbody>
</table>

Definition at line 179 of file BaseForecaster.cxx.

48.2.2.3  void Verdandi::BaseForecaster::Aggregate ( int t, const Matrix<T> & simulation_data, const Vector<T> & observation_data, const Vector3<T> & ensemble, const Vector2<T> & observation, Matrix<T> & weight, Vector2<T> & aggregated ) [virtual]

Aggregates at a given time.

It carries out the sequential aggregation and returns the aggregated forecast together with the aggregation weights. This method only carries out the aggregation at one time.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>t</strong></td>
<td>time index at which the aggregation is carried out.</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>simulation_data</strong></td>
<td>the ensemble data to be aggregated indexed by member and location.</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>observation_data</strong></td>
<td>the observational data, indexed by location.</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>ensemble</strong></td>
<td>the ensemble data indexed by member, time and location.</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>observation</strong></td>
<td>the observational data, indexed by time and location.</td>
</tr>
<tr>
<td><strong>out</strong></td>
<td><strong>weight</strong></td>
<td>aggregation weights, indexed by member and time.</td>
</tr>
<tr>
<td><strong>out</strong></td>
<td><strong>aggregated</strong></td>
<td>the aggregated output, indexed by time and member.</td>
</tr>
</tbody>
</table>

Definition at line 547 of file BaseForecaster.cxx.

48.2.2.4  void Verdandi::BaseForecaster::AggregateMean ( int t, const Matrix<T> & simulation_data, const Vector3<T> & ensemble, Matrix<T> & weight, Vector2<T> & aggregated )

Computes the ensemble mean at a given time.

It carries out the aggregation and returns the aggregated forecast together with the aggregation weights. This method only carries out the aggregation at one time.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td><strong>t</strong></td>
<td>time index at which the aggregation is carried out.</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>simulation_data</strong></td>
<td>the ensemble data to be aggregated indexed by member and location.</td>
</tr>
<tr>
<td><strong>in</strong></td>
<td><strong>ensemble</strong></td>
<td>the ensemble data indexed by member, time and location.</td>
</tr>
</tbody>
</table>
Verdandi::BaseForecaster Class Reference

48.2.5 void Verdandi::BaseForecaster::AggregateSpinUpPeriod ( int t, const Matrix<T> & simulation_data, const Vector<T> & observation_data, const Vector3<T> & ensemble, const Vector2<T> & observation, Matrix<T> & weight, Vector2<T> & aggregated ) [virtual]

Aggregates in the spin-up period at a given time.
It carries out the sequential aggregation and returns the aggregated forecast together with the aggregation weights. This method only carries out the aggregation at one time.

Parameters

| in | t | time index at which the aggregation is carried out. |
|    | simulation_data | the ensemble data to be aggregated indexed by member and location. |
|    | observation_data | the observational data, indexed by location. |
|    | ensemble | the ensemble data indexed by member, time and location. |
|    | observation | the observational data, indexed by time and location. |
|    | weight | aggregation weights, indexed by member and time. |
| out | aggregated | the aggregated output, indexed by time and member. |

Reimplemented in Verdandi::DiscountedRidgeRegression.

Definition at line 335 of file BaseForecaster.cxx.

48.2.6 void Verdandi::BaseForecaster::AggregateSpinUpPeriodThreshold ( int t, const Matrix<T> & simulation_data, const Vector<T> & observation_data, const Vector3<T> & ensemble, const Vector2<T> & observation, T & threshold_value, Matrix<T> & weight, Matrix<T> & base_weight, Vector2<T> & aggregated ) [virtual]

Aggregates in the spin-up period at a given time.
It carries out the sequential aggregation with a threshold on the weights and returns the aggregated forecast together with the aggregation weights. This method only carries out the aggregation at one time.

Parameters

| in | t | time index at which the aggregation is carried out. |
|    | simulation_data | the ensemble data to be aggregated indexed by member and location. |
|    | observation_data | the observational data, indexed by location. |
|    | ensemble | the ensemble data indexed by member, time and location. |
|    | observation | the observational data, indexed by time and location. |
|    | threshold_value | threshold on the weights. |
| out | weight | aggregation weights, indexed by member and time. |
| out | base_weight | aggregation weights before the threshold is applied, indexed by member and time. |
| out | aggregated | the aggregated output, indexed by time and member. |

Definition at line 369 of file BaseForecaster.cxx.
48.2.7  void Verdandi::BaseForecaster::AggregateThreshold ( const Vector3<T> & ensemble, const Vector2<T> & observation, T threshold_value, Vector2<T> & aggregated ) [virtual]

Aggregates the ensemble data.
It carries out the sequential aggregation with a threshold on the weights and returns the aggregated forecast.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ensemble</th>
<th>the ensemble data indexed by member, time and location.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>observation</td>
<td>the observational data, indexed by time and location.</td>
</tr>
<tr>
<td>in</td>
<td>threshold_value</td>
<td>threshold on the weights.</td>
</tr>
<tr>
<td>out</td>
<td>aggregated</td>
<td>the aggregated output, indexed by time and location.</td>
</tr>
</tbody>
</table>

Definition at line 155 of file BaseForecaster.cxx.

48.2.8  void Verdandi::BaseForecaster::AggregateThreshold ( const Vector3<T> & ensemble, const Vector2<T> & observation, T threshold_value, Matrix<T> & weight, Vector2<T> & aggregated ) [virtual]

Aggregates the ensemble data.
It carries out the sequential aggregation with a threshold on the weights and returns the aggregated forecast together with the aggregation weights.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ensemble</th>
<th>the ensemble data indexed by member, time and location.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>observation</td>
<td>the observational data, indexed by time and location.</td>
</tr>
<tr>
<td>in</td>
<td>threshold_value</td>
<td>threshold on the weights.</td>
</tr>
<tr>
<td>out</td>
<td>weight</td>
<td>aggregation weights, indexed by member and time.</td>
</tr>
<tr>
<td>out</td>
<td>aggregated</td>
<td>the aggregated output, indexed by time and member.</td>
</tr>
</tbody>
</table>

Definition at line 235 of file BaseForecaster.cxx.

48.2.9  void Verdandi::BaseForecaster::AggregateThreshold ( int t, const Matrix<T> & simulation_data, const Vector<T> & observation_data, const Vector3<T> & ensemble, const Vector2<T> & observation, T & threshold_value, Matrix<T> & weight, Matrix<T> & base_weight, Vector2<T> & aggregated ) [virtual]

Aggregates at a given time.
It carries out the sequential aggregation with a threshold on the weights and returns the aggregated forecast together with the aggregation weights. This method only carries out the aggregation at one time.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>t</th>
<th>time index at which the aggregation is carried out.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>simulation_data</td>
<td>the ensemble data to be aggregated indexed by member and location.</td>
</tr>
<tr>
<td>in</td>
<td>observation_data</td>
<td>the observational data, indexed by location.</td>
</tr>
<tr>
<td>in</td>
<td>ensemble</td>
<td>the ensemble data indexed by member, time and location.</td>
</tr>
<tr>
<td>in</td>
<td>observation</td>
<td>the observational data, indexed by time and location.</td>
</tr>
<tr>
<td>in</td>
<td>threshold_value</td>
<td>threshold on the weights.</td>
</tr>
<tr>
<td>out</td>
<td>weight</td>
<td>aggregation weights, indexed by member and time.</td>
</tr>
<tr>
<td>out</td>
<td>base_weight</td>
<td>aggregation weights before the threshold is applied, indexed by member and time.</td>
</tr>
<tr>
<td>out</td>
<td>aggregated</td>
<td>the aggregated output, indexed by time and member.</td>
</tr>
</tbody>
</table>

Definition at line 589 of file BaseForecaster.cxx.
48.2.2.10 void Verdandi::BaseForecaster::AggregateZero ( int t, const Vector3 & ensemble, Matrix & weight, Vector2 & aggregated )

Aggregates with zero weights at a given time.

It carries out the aggregation and returns the aggregated forecast together with the aggregation weights. This method only carries out the aggregation at one time.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>int</td>
</tr>
<tr>
<td>in</td>
<td>simulation_data</td>
</tr>
<tr>
<td>in</td>
<td>observation_data</td>
</tr>
<tr>
<td>in</td>
<td>ensemble</td>
</tr>
<tr>
<td>out</td>
<td>weight</td>
</tr>
<tr>
<td>out</td>
<td>aggregated</td>
</tr>
</tbody>
</table>

Definition at line 686 of file BaseForecaster.cxx.

48.2.2.11 void Verdandi::BaseForecaster::ComputeAggregatedValue ( int t, const Vector3 & ensemble, const Vector & weight_vector, Vector2 & aggregated ) [virtual]

Computes the aggregated value based on given weights.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>int</td>
</tr>
<tr>
<td>in</td>
<td>ensemble</td>
</tr>
<tr>
<td>in</td>
<td>weight_vector</td>
</tr>
<tr>
<td>out</td>
<td>aggregated</td>
</tr>
</tbody>
</table>

Definition at line 512 of file BaseForecaster.cxx.

48.2.2.12 void Verdandi::BaseForecaster::ComputeWeight ( int t, const Matrix & simulation_data, const Vector & observation_data, const Vector3 & ensemble, const Vector2 & observation, const Matrix & weight, Vector & weight_vector ) [virtual]

Computes aggregation weights at a given time.

It computes the aggregation weights at one time. This method updates the weights at time $t$ based on past weights, which may be more efficient than computing the weights from scratch.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>int</td>
</tr>
<tr>
<td>in</td>
<td>simulation_data</td>
</tr>
<tr>
<td>in</td>
<td>observation_data</td>
</tr>
<tr>
<td>in</td>
<td>ensemble</td>
</tr>
<tr>
<td>in</td>
<td>observation</td>
</tr>
<tr>
<td>out</td>
<td>weight</td>
</tr>
<tr>
<td>out</td>
<td>weight_vector</td>
</tr>
</tbody>
</table>

Reimplemented in Verdandi::DiscountedRidgeRegression.

Definition at line 431 of file BaseForecaster.cxx.
void Verdandi::BaseForecaster::ComputeWeightNonSequential ( int t, const Vector3< T >& ensemble, const Vector2< T >& observation, Vector< T >& weight_vector ) [virtual]

Computes aggregation weights at a given time.

It computes the aggregation weights at one time. This method computes the weights at time \( t \) without knowledge of the past weights, which may be less efficient than relying on past weights.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>t</th>
<th>time index at which the aggregation is carried out.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>ensemble</td>
<td>the ensemble data indexed by member, time and location.</td>
</tr>
<tr>
<td>in</td>
<td>observation</td>
<td>the observational data, indexed by time and location.</td>
</tr>
<tr>
<td>out</td>
<td>weight_vector</td>
<td>aggregation weights for time ( t ).</td>
</tr>
</tbody>
</table>

Reimplemented in Verdandi::DiscountedRidgeRegression.

Definition at line 458 of file BaseForecaster.cxx.

int Verdandi::BaseForecaster::GetNparameter ( ) const [inline]

Return the number of parameters.

Returns

The number of parameters in the algorithm.

Definition at line 81 of file BaseForecaster.cxx.

int Verdandi::BaseForecaster::GetNspinup ( ) const [inline]

Returns the width of the spin-up period.

Returns

The width of the spin-up period.

Definition at line 59 of file BaseForecaster.cxx.

const Vector< T >& Verdandi::BaseForecaster::GetParameter ( ) const [inline]

Returns the vector of parameters.

Returns

The vector of parameters.

Definition at line 93 of file BaseForecaster.cxx.

void Verdandi::BaseForecaster::Init ( const Vector3< T >& ensemble, const Vector2< T >& observation ) [virtual]

Prepares the aggregation.

This method should be called before an aggregation method is called.

Parameters
Reimplemented in Verdandi::DiscountedRidgeRegression.
Definition at line 301 of file BaseForecaster.cxx.

### 48.2.2.18 bool Verdandi::BaseForecaster::IsConvex ( ) const [inline]
Is the method carrying out convex aggregation?
It returns true if the methods produces convex combinations.

Returns
true if the method is convex, false otherwise.
Definition at line 115 of file BaseForecaster.cxx.

### 48.2.2.19 void Verdandi::BaseForecaster::SetNspinup ( int Nspinup )
Sets the width of the spin-up period.

Parameters
- **in** `Nspinup` new width of the spin-up period.

Definition at line 70 of file BaseForecaster.cxx.

### 48.2.2.20 void Verdandi::BaseForecaster::SetParameter ( const Vector< T > & parameter ) [virtual]
Sets the vector of parameters.

Parameters
- **in** `parameter` new vector of parameters.

Reimplemented in Verdandi::DiscountedRidgeRegression.
Definition at line 104 of file BaseForecaster.cxx.

The documentation for this class was generated from the following files:

- `/nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/sequential_aggregation/BaseForecaster.hxx`
- `/nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/sequential_aggregation/BaseForecaster.cxx`

---

### 48.3 Verdandi::BasePerturbationManager Class Reference

This class generates and applies perturbations.

```cpp
#include <BasePerturbationManager.hxx>
```

Inheritance diagram for Verdandi::BasePerturbationManager:
Public Member Functions

- `BasePerturbationManager ()`
  Default constructor.
- `BasePerturbationManager (string configuration_file)`
  Main constructor.
- `~BasePerturbationManager ()`
  Destructor.
- `void Initialize (string configuration_file)`
  Initializes the manager.
- `template<class T0 , class Prop0 , class Allocator0 , class T1 , class Allocator1 > void Sample (string pdf, Matrix<T0, Prop0, RowSymPacked, Allocator0> &variance, Vector<double, VecFull> &parameter, Vector<double, VecFull> &correlation, Vector<T1, VecFull, Allocator1> &output)`
  Generates a random vector or several vectors according to some distribution.
- `template<class T0 , class Prop0 , class Allocator0 , class T1 , class Allocator1 > void Sample (string pdf, Matrix<T0, Prop0, RowSymPacked, Allocator0> &variance, Vector<double, VecFull> &parameter, Vector<double, VecFull> &correlation, Vector<T1, Collection, Allocator1> &output)`
  Generates a random vector collection according to some distribution.
- `template<class T0 , class T1 , class Allocator1 > void Sample (string pdf, T0 variance, Vector<double, VecFull> &parameter, Vector<double, VecFull> &correlation, Vector<T1, VecFull, Allocator1> &output)`
  Generates a random vector or several vectors according to a homogeneous distribution.
- `template<class T0 , class T1 , class Allocator1 > void Sample (string pdf, T0 variance, Vector<double, VecFull> &parameter, Vector<double, VecFull> &correlation, Vector<T1, Collection, Allocator1> &output)`
  Generates a random vector collection according to a homogeneous distribution.
- `virtual string GetName () const`
  Returns the name of the class.
- `virtual void Message (string message)`
  Receives and handles a message.

Static Public Member Functions

- `static void StaticMessage (void *object, string message)`
  Receives and handles a message with a static method.

48.3.1 Detailed Description

This class generates and applies perturbations.
48.3.2 Constructor & Destructor Documentation

48.3.2.1 Verdandi::BasePerturbationManager::BasePerturbationManager ( )

Default constructor.
Builds the manager.
Definition at line 42 of file BasePerturbationManager.cxx.

48.3.2.2 Verdandi::BasePerturbationManager::BasePerturbationManager ( string configuration_file )

Main constructor.
Builds the manager.

Parameters

| in | configuration_file | configuration file |

Definition at line 57 of file BasePerturbationManager.cxx.

48.3.3 Member Function Documentation

48.3.3.1 string Verdandi::VerdandiBase::GetName ( ) const [virtual, inherited]

Returns the name of the class.

Returns

The name of the class.

Reimplemented in Verdandi::LinearObservationManager, Verdandi::PetscLinearObservationManager, Verdandi::GridToNetworkObservationManager, Verdandi::ShallowWater, Verdandi::ClampedBar, Verdandi::PetscClampedBar, Verdandi::LevelSetObservationManager, Verdandi::QuadraticModel, Verdandi::ReducedOrderUnscentedKalmanFilter, Verdandi::ReducedMinimax, Verdandi::PythonModel, Verdandi::ReducedOrderExtendedKalmanFilter, Verdandi::HamiltonJacobiBellman, Verdandi::UnscentedKalmanFilter, Verdandi::EnsembleKalmanFilter, Verdandi::FourDimensionalVariational, Verdandi::PythonObservationManager, Verdandi::OptimalInterpolation, Verdandi::ExtendedKalmanFilter, Verdandi::ExtendedMinimaxFilter, Verdandi::FrontPositionObserver, Verdandi::ModelTemplate, Verdandi::MonteCarlo, Verdandi::ObservationManagerTemplate, Verdandi::Lorenz, Verdandi::Nudging, Verdandi::ObservationGenerator, Verdandi::ForwardDriver, and Verdandi::CheckingModel.

Definition at line 51 of file VerdandiBase.cxx.

48.3.3.2 void Verdandi::BasePerturbationManager::Initialize ( string configuration_file )

Initializes the manager.

Parameters

| in | configuration_file | configuration file |

Reimplemented in Verdandi::TR1PerturbationManager, Verdandi::NewranPerturbationManager, Verdandi::RandomPerturbationManager, and Verdandi::TRNGPerturbationManager.

Definition at line 82 of file BasePerturbationManager.cxx.
Generates a random vector or several vectors according to some distribution.

This method generates one vector or several vectors, according to the shape of output on entry. If variance is a $N \times N$ matrix, then output must be a vector of size $mN$, and this method generates a sample of $m$ vectors of size $N$, stored in output. The sample can be generated with or without correlation between the vectors, depending on correlation.

**Parameters**

- **pdf** probability density function: "Normal" or "LogNormal".
- **variance** covariance matrix of the distribution.
- **parameter** vector of parameters. The vector may either be empty or contain two clipping parameters $(a, b)$. With the clipping parameters, for a normal (log-normal) distribution, any component $i$ of the vector (the logarithm of the vector) lies in $[\mu_i - a\sigma_i, \mu_i + b\sigma_i]$ where $\mu_i$ is the mean (median) of the random component $i$ and $\sigma_i$ is its standard deviation (the standard deviation of its logarithm).
- **correlation** if non-empty, correlations with the first vector: correlation should then contain $m - 1$ elements if there are $m$ vectors to be generated.
- **output** on entry, mean or median vector(s); on exit, the sample. If the size of output is not a multiple of $N$, an exception is thrown.

**Definition at line 116 of file BasePerturbationManager.cxx.**

Generates a random vector collection according to some distribution.

**Parameters**

- **pdf** probability density function: "Normal" or "LogNormal".
- **variance** covariance matrix of the distribution.
- **parameter** vector of parameters. The vector may either be empty or contain two clipping parameters $(a, b)$. With the clipping parameters, for a normal (log-normal) distribution, any component $i$ of the vector (the logarithm of the vector) lies in $[\mu_i - a\sigma_i, \mu_i + b\sigma_i]$ where $\mu_i$ is the mean (median) of the random component $i$ and $\sigma_i$ is its standard deviation (the standard deviation of its logarithm).
- **correlation** if non-empty, correlations with the first vector: correlation should then contain $m - 1$ elements if there are $m$ vectors to be generated.
- **output** on entry, mean or median vectors; on exit, the sample in the form of a vector collection.

**Definition at line 260 of file BasePerturbationManager.cxx.**

Generates a random vector or several vectors according to a homogeneous distribution.

This method generates one vector or several vectors, according to the shape of output on entry. If correlation is
of size $m - 1$, then `output` must be a vector of size $mN$, and this method generates a sample of $m$ vectors of size $N$, stored in `output`. The sample can be generated with or without correlation between the vectors, depending on `correlation`.

For a normal (log-normal) homogeneous distribution, for each vector of size $N$, the same random number, with centered normal distribution and variance `variance`, is added to every component of the vector (logarithm of the vector).

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th><code>pdf</code></th>
<th>probability density function: &quot;NormalHomogeneous&quot; or &quot;LogNormal-Homogeneous&quot;.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>variance</code></td>
<td>variance of the distribution.</td>
</tr>
<tr>
<td>in</td>
<td><code>parameter</code></td>
<td>vector of parameters. The vector may either be empty or contain two clipping parameters $(a, b)$. With the clipping parameters, for a normal (log-normal) distribution, any random value lies in $[\mu - a\sigma, \mu + b\sigma]$ where $\mu$ is the mean (median) of the random variable and $\sigma$ is its standard deviation (the standard deviation of its logarithm).</td>
</tr>
<tr>
<td>in</td>
<td><code>correlation</code></td>
<td>if non-empty, correlations with the first vector: <code>correlation</code> should then contain $m - 1$ elements if there are $m$ vectors to be sampled.</td>
</tr>
<tr>
<td>in,out</td>
<td><code>output</code></td>
<td>on entry, mean or median vector(s); on exit, the sample. If the size of <code>output</code> is not a multiple of that of <code>correlation</code> plus one, an exception is thrown.</td>
</tr>
</tbody>
</table>

Definition at line 430 of file `BasePerturbationManager.cxx`.

#### 48.3.3.6 template <class T0 , class T1 , class Allocator1 > void Verdandi::BasePerturbationManager::Sample ( string `pdf`, T0 `variance`, Vector< double, VectFull > & `parameter`, Vector< double, VectFull > & `correlation`, Vector< T1, Collection, Allocator1 > & `output` )

Generates a random vector collection according to a homogeneous distribution.

The number of vectors in `output` should be the size of `correlation` plus one.

For a normal (log-normal) homogeneous distribution, for each vector, the same random number, with centered normal distribution and variance `variance`, is added to every component of the vector (logarithm of the vector).

### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th><code>pdf</code></th>
<th>probability density function: &quot;NormalHomogeneous&quot; or &quot;LogNormal-Homogeneous&quot;.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td><code>variance</code></td>
<td>variance of the distribution.</td>
</tr>
<tr>
<td>in</td>
<td><code>parameter</code></td>
<td>vector of parameters. The vector may either be empty or contain two clipping parameters $(a, b)$. With the clipping parameters, for a normal (log-normal) distribution, any random value lies in $[\mu - a\sigma, \mu + b\sigma]$ where $\mu$ is the mean (median) of the random variable and $\sigma$ is its standard deviation (the standard deviation of its logarithm).</td>
</tr>
<tr>
<td>in</td>
<td><code>correlation</code></td>
<td>if non-empty, correlations with the first vector: <code>correlation</code> should then contain $m - 1$ elements if there are $m$ vectors to be generated.</td>
</tr>
<tr>
<td>in,out</td>
<td><code>output</code></td>
<td>output on entry, mean or median vectors of the vector collection; on exit, the sample in the form of a vector collection.</td>
</tr>
</tbody>
</table>

Definition at line 547 of file `BasePerturbationManager.cxx`.

The documentation for this class was generated from the following files:

- `/nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/BasePerturbationManager.hxx`
- `/nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/BasePerturbationManager.cxx`
48.4 Verdandi::CheckingModel Class Reference

This class is a model template.

```cpp
#include <CheckingModel.hxx>
```

Inheritance diagram for Verdandi::CheckingModel:

```
Verdandi::VerdandiBase

Verdandi::CheckingModel
```

**Public Types**

- `typedef Model::state_error_variance_row state_error_variance_row` 
  Type of a row of the background error variance.
- `typedef Model::state state` 
  Type of the model state vector.
- `typedef Model::state_error_variance state_error_variance` 
  Type of the background error variance.
- `typedef Model::error_variance error_variance` 
  Type of the model error variance.
- `typedef Model::tangent_linear_operator tangent_linear_operator` 
  Type of the tangent linear model.
- `typedef Model::matrix_state_observation matrix_state_observation` 
  Type of the model/observation crossed matrix.
- `typedef Model::state_error_variance_reduced state_error_variance_reduced` 
  Type of the reduced matrix $U$ in the $LU^T$ decomposition of the background error covariance matrix.

**Public Member Functions**

- `CheckingModel ()` 
  Constructor.
- `~CheckingModel ()` 
  Destructor.
- `void Initialize (string configuration_file)` 
  Initializes the model.
- `void InitializeFirstStep ()` 
  Initializes the current time step for the model.
- `void InitializeStep ()` 
  Initializes the current time step for the model.
- `void Forward ()` 
  Advances one step forward in time.
- `void BackwardAdjoint (state &observation_term)` 
  Performs one step backward in adjoint model.
- `bool HasFinished () const` 
  Checks whether the model has finished.
- `void FinalizeStep ()` 
  Finalizes the current time step for the model.
- `void Finalize ()`
Finalizes the model.

- double ApplyOperator (state &x, bool preserve_state=true)
  Applies the model to a given vector.

- double ApplyTangentLinearOperator (state &x)
  Applies the tangent linear model to a given vector.

- tangent_linear_operator& GetTangentLinearOperator ()
  Gets the tangent linear model.

- double GetTime ()
  Returns the current time.

- void SetTime (double time)
  Sets the time of the model to a given time.

- int GetNstate ()
  Returns the state vector size.

- int GetNfull_state ()
  Returns the size of the full state vector.

- state & GetState ()
  Provides the state vector.

- void StateUpdated ()
  Sets the state vector.

- state & GetFullState ()
  Provides the full state vector.

- void FullStateUpdated ()
  Sets the full state vector.

- state & GetStateLowerBound ()
  Provides the state lower bound.

- state & GetStateUpperBound ()
  Provides the state upper bound.

- state & GetAdjointState ()
  Returns the adjoint state vector.

- void AdjointStateUpdated ()
  Sets the adjoint state vector.

- state_error_variance_row & GetStateErrorVarianceRow (int row)
  Computes a row of the variance of the state error.

- state_error_variance & GetStateErrorVariance ()
  Returns the state error variance.

- state_error_variance & GetStateErrorVarianceProjector ()

- state_error_variance_reduced & GetStateErrorVarianceReduced ()

- const state_error_variance & GetStateErrorVarianceInverse () const

- error_variance & GetErrorVariance ()
  Returns the model error variance.

- string GetName () const
  Returns the name of the class.

- void Message (string message)
  Receives and handles a message.

Static Public Member Functions

- static void StaticMessage (void *object, string message)
  Receives and handles a message with a static method.
Public Attributes

- Model model_
  The model to be tested.

48.4.1 Detailed Description

This class is a model template.

48.4.2 Member Function Documentation

48.4.2.1 void Verdandi::CheckingModel::AdjointStateUpdated()

Sets the adjoint state vector.

Parameters

| out | state_adjoint | the adjoint state vector. |

Definition at line 443 of file CheckingModel.cxx.

48.4.2.2 double Verdandi::CheckingModel::ApplyOperator (state & x, bool preserve_state = true)

Applies the model to a given vector.

The current state of the model is modified.

Parameters

| in | x | a vector. |
| in | preserve_state | Boolean to indicate if the model state has to be preserved. |

Definition at line 148 of file CheckingModel.cxx.

48.4.2.3 double Verdandi::CheckingModel::ApplyTangentLinearOperator (state & x)

Applies the tangent linear model to a given vector.

Parameters

| in | x | a vector. |

Definition at line 206 of file CheckingModel.cxx.

48.4.2.4 void Verdandi::CheckingModel::BackwardAdjoint (state & observation_term)

Performs one step backward in adjoint model.

Parameters

| in | observation_-term | $H^T R^{-1} (y - H x)$. |

Definition at line 103 of file CheckingModel.cxx.
48.4.2.5  void Verdandi::CheckingModel::Forward ( )
Advances one step forward in time.

\[ x_{n+1} = M_h(x_n, p_0). \]
Definition at line 92 of file CheckingModel.cxx.

48.4.2.6  void Verdandi::CheckingModel::FullStateUpdated ( )
Sets the full state vector.
Parameters

| in | state | the full state vector. |
Definition at line 420 of file CheckingModel.cxx.

48.4.2.7  CheckingModel<Model> & Verdandi::CheckingModel::GetAdjointState ( )
Returns the adjoint state vector.
Parameters

| out | state_adjoint | the adjoint state vector. |
Definition at line 432 of file CheckingModel.cxx.

48.4.2.8  CheckingModel<Model> &error_variance & Verdandi::CheckingModel::GetErrorVariance ( )
Returns the model error variance.

Returns

The model error variance.

Note
If the matrix is empty, it is then assumed there is no model error.
Definition at line 547 of file CheckingModel.cxx.

48.4.2.9  CheckingModel<Model> &state & Verdandi::CheckingModel::GetFullState ( )
Provides the full state vector.
Parameters

| out | state | the full state vector. |
Definition at line 409 of file CheckingModel.cxx.

48.4.2.10 string Verdandi::CheckingModel::GetName ( ) const [virtual]
Returns the name of the class.
Returns

The name of the class.

Reimplemented from Verdandi::VerdandiBase.
Definition at line 558 of file CheckingModel.cxx.

48.4.2.11 int Verdandi::CheckingModel::GetNfull_state ( )

Returns the size of the full state vector.

Returns

The size of the full state vector.

Definition at line 342 of file CheckingModel.cxx.

48.4.2.12 int Verdandi::CheckingModel::GetNstate ( )

Returns the state vector size.

Returns

The state vector size.

Definition at line 324 of file CheckingModel.cxx.

48.4.2.13 CheckingModel< Model >::state & Verdandi::CheckingModel::GetState ( )

Provides the state vector.

Parameters

| out | state | the reduced state vector. |

Definition at line 362 of file CheckingModel.cxx.

48.4.2.14 CheckingModel< Model >::state_error_variance & Verdandi::CheckingModel::GetStateErrorVariance ( )

Returns the state error variance.

Returns

The state error variance.

Definition at line 473 of file CheckingModel.cxx.

48.4.2.15 CheckingModel< Model >::state_error_variance & Verdandi::CheckingModel::GetStateErrorVarianceProjector ( )

Returns the matrix L in the decomposition of the state error covariance matrix (B) as a product \( LUL^T \).

Returns

The matrix \( L \).

Definition at line 487 of file CheckingModel.cxx.
48.4.2.16 CheckingModel::state_error_variance_reduced & Verdandi::CheckingModel::GetStateErrorVarianceReduced ( )

Returns the matrix $U$ in the decomposition of the state error covariance matrix ($B$) as a product $LUU^T$.

Returns

The matrix $U$.

Definition at line 522 of file CheckingModel.cxx.

48.4.2.17 CheckingModel::state_error_variance_row & Verdandi::CheckingModel::GetStateErrorVarianceRow ( int row )

Computes a row of the variance of the state error.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>row</th>
<th>row index.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>$P_{row}$</td>
<td>the row with index row in the state error variance.</td>
</tr>
</tbody>
</table>

Definition at line 461 of file CheckingModel.cxx.

48.4.2.18 CheckingModel::state & Verdandi::CheckingModel::GetStateLowerBound ( )

Provides the state lower bound.

Parameters

| out | lower_bound | the state lower bound (componentwise). |

Definition at line 386 of file CheckingModel.cxx.

48.4.2.19 CheckingModel::state & Verdandi::CheckingModel::GetStateUpperBound ( )

Provides the state upper bound.

Parameters

| out | upper_bound | the state upper bound (componentwise). |

Definition at line 398 of file CheckingModel.cxx.

48.4.2.20 CheckingModel::tangent_linear_operator & Verdandi::CheckingModel::GetTangentLinearOperator ( )

Gets the tangent linear model.

Parameters

| out | $A$ | the matrix of the tangent linear model. |

Definition at line 280 of file CheckingModel.cxx.
double Verdandi::CheckingModel::GetTime() const

Returns the current time.

Definition at line 296 of file CheckingModel.cxx.

bool Verdandi::CheckingModel::HasFinished() const

Checks whether the model has finished.

Definition at line 114 of file CheckingModel.cxx.

void Verdandi::CheckingModel::Initialize(string configuration_file)

Initializes the model.

Parameters

<table>
<thead>
<tr>
<th>configuration_file</th>
<th>configuration file.</th>
</tr>
</thead>
</table>

Definition at line 62 of file CheckingModel.cxx.

void Verdandi::CheckingModel::SetTime(double time)

Sets the time of the model to a given time.

Parameters

<table>
<thead>
<tr>
<th>time</th>
<th>a given time.</th>
</tr>
</thead>
</table>

Definition at line 307 of file CheckingModel.cxx.

void Verdandi::CheckingModel::StateUpdated()

Sets the state vector.

Before setting the reduced state vector, special requirements can be enforced; e.g. positivity requirement or inferior and superior limits.

Parameters

<table>
<thead>
<tr>
<th>state</th>
<th>the reduced state vector.</th>
</tr>
</thead>
</table>

Definition at line 374 of file CheckingModel.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/model/CheckingModel.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/model/CheckingModel.cxx
48.5 Verdandi::ClampedBar Class Reference

This class is a clamped-bar model.

#include <ClampedBar.hxx>

Inheritance diagram for Verdandi::ClampedBar:

```
Verdandi::VerdandiBase
   |   |
   V   V
Verdandi::ClampedBar
```

Public Types

- `typedef T value_type`
  *The numerical type (e.g., double).*
- `typedef T * pointer`
  *Pointer to the numerical type.*
- `typedef const T * const_pointer`
  *Const pointer to the numerical type.*
- `typedef T & reference`
  *Reference to the numerical type.*
- `typedef const T & const_reference`
  *Const reference to the numerical type.*
- `typedef Matrix< T > state_error_variance`
  *Type of the background error covariance matrix.*
- `typedef Matrix< T > state_error_variance_reduced`
  *Type of the reduced matrix \( U \) in the \( L U L^T \) decomposition of the background error covariance matrix.*
- `typedef Vector< T > state_error_variance_row`
  *Type of a row of the background error variance.*
- `typedef Matrix< T > state_error_variance_row_matrix_error_variance`
  *Type of the model error variance.*
- `typedef Matrix< T > matrix_state_observation`
  *Type of the model/observation crossed matrix.*
- `typedef Matrix< T > tangent_linear_operator`
  *Type of the tangent linear operator.*
- `typedef Vector< T > state`
  *Type of the model state vector.*
- `typedef Vector< state, Collection > state_collection`
  *Collection of vector state.*

Public Member Functions

- `ClampedBar ()`
  *Constructor.*
- `ClampedBar (string configuration_file)`
  *Constructor.*
- `~ClampedBar ()`
  *Destructor.*
- `void Initialize (string configuration_file)`

Generated on Mon Sep 28 2015 16:49:58 for Verdandi by Doxygen
Initializes the model.
- void InitializeStep()
  Initializes the current time step for the model.
- void InitializeAdjoint()
  Initializes the adjoint model.
- void Forward(bool update_force=true)
  Advances one step forward in time.
- bool HasFinished() const
  Checks whether the model has finished.
- void FinalizeStep()
  Finalizes the current time step for the model.
- void Finalize()
  Finalizes the model.
- void Save()
  Saves the simulated data.
- void BackwardAdjoint(state &state_innovation)
  Performs one step backward in adjoint model.
- double ApplyOperator(state &x, bool preserve_state=true, bool update_force=true)
  Applies the model to a given state vector.
- double ApplyTangentLinearOperator(state &x)
  Applies the tangent linear model to a given vector.
- tangent_linear_operator& GetTangentLinearOperator()
  Gets the matrix of the tangent linear model.
- double GetTime() const
  Returns the current time.
- void SetTime(double time)
  Sets the time of the model to a given time.
- int GetNstate() const
  Returns the state vector size.
- void GetStateCopy(state &state)
  Provides the reduced state vector.
- void SetStateCopy(state &state)
  Sets the reduced state vector.
- state & GetState()
  Provides the reduced state vector.
- void StateUpdated()
  Performs some calculations when the update of the model state is done.
- state & GetStateLowerBound()
  Provides the state lower bound.
- state & GetStateUpperBound()
  Provides the state upper bound.
- state & GetFullState()
  Provides the full state vector.
- void FullStateUpdated()
  Performs some calculations when the update of the model state is done.
- state & GetAdjointState()
  Returns the adjoint state vector.
- void AdjointStateUpdated()
- state & GetAdditionalAdjointTerm()
  Returns a term that will be added to the adjoint state.
- double GetAdditionalCostTerm()
Verdandi::ClampedBar Class Reference

Returns a term that will be added to the cost function.

- `state_error_variance_row & GetStateErrorVarianceRow (int row)`
  Computes a row of the background error covariance matrix B.
- `state_error_variance & GetStateErrorVariance ()`
  Returns the background error covariance matrix (B).
- `const state_error_variance & GetStateErrorVariance () const`
  Returns the background error covariance matrix (B).
- `state_error_variance & GetStateErrorVarianceProjector ()`
- `state_error_variance_reduced & GetStateErrorVarianceReduced ()`
- `const state_error_variance & GetStateErrorVarianceInverse () const`
  Returns the inverse of the background error variance (B\(^{-1}\)).
- `error_variance & GetErrorVariance ()`
  Returns the model error variance.
- `string GetName () const`
  Returns the name of the class.
- `void Message (string message)`
  Receives and handles a message.

**Static Public Member Functions**

- `static void StaticMessage (void *object, string message)`
  Receives and handles a message with a static method.

**Protected Attributes**

- `double bar_length_`
  Bar length.
- `double Delta_x_`
  Space step along x.
- `int Nx_`
  Number of elements along x.
- `int Ndof_`
  Number of degrees of freedom (dofs).
- `int Nstate_`
  Size of the state vector.
- `double Delta_t_`
  Time step.
- `double time_`
  Current time.
- `double final_time_`
  Simulation duration.
- `double mass_density_`
  Mass parameter.
- `double Young_modulus_`
  Young’s Modulus.
- `Vector<T> theta_force_`
  Force parameter.
- `int Ntheta_force_`
  Number of force parameter regions.
- `Vector<int> theta_force_index_`
Force parameter region of elements.

- Vector<T> theta_stiffness
  Stiffness parameter.
- int Ntheta_stiffness
  Number of stiffness parameter regions.
- Vector<int> theta_stiffness_index
  Stiffness parameter region of elements.
- Vector<T> theta_damp
  Damp parameter.
- int Ntheta_damp
  Number of damp parameter regions.
- Vector<int> theta_damp_index
  Damp parameter region of elements.
- state_collection x
  State collection.
- state_collection x_full
  Full state collection.
- state lower_bound
  State lower bound.
- state upper_bound
  State upper bound.
- Vector<T> theta_mass
  Mass parameter.
- int Ntheta_mass
  Number of mass parameter regions.
- Vector<int> theta_mass_index
  Mass parameter region of elements.
- state disp_0
  FEM Vector (disp 0).
- state velo_0
  FEM Vector (velo 0).
- state force
  FEM Vector (force).
- set<string> stable
  State.
- vector<string> reduced
  Reduced state.
- Matrix<T, General, RowMajor> mass_FEM_matrix
  Newmark Global FEM matrix (mass matrix).
- Matrix<T, General, RowMajor> stiffness_FEM_matrix
  Newmark Global FEM matrix (Newmark matrix 0).
- Matrix<T, General, RowMajor> damp_FEM_matrix
  Newmark Global FEM matrix (Newmark matrix 1).
- Matrix<T, Symmetric, RowSymSparse> mass_matrix
  Damp matrix (C).
- double alpha
  Damp alpha coefficient.
- double beta
Damp beta coefficient.

- Matrix< T, Symmetric, RowSymSparse > stiffness_matrix_
  Stiffness matrix (K).
- double state_error_variance_value_
  Background error variance.
- state_error_variance state_error_variance_
  Background error covariance matrix (B).
- state_error_variance state_error_variance_inverse_
  Inverse of the background error covariance matrix (B⁻¹).
- state_error_variance state_error_variance_projector_
  Projector matrix L in the decomposition of the background error covariance matrix (B) as a product LULᵀ.
- state_error_variance_reduced state_error_variance_reduced_
  Reduced matrix U in the decomposition of the background error covariance matrix (B) as a product LULᵀ.
- bool variance_projector_allocated_
  Is state error variance projector allocated?
- bool variance_reduced_allocated_
  Is reduced state error variance allocated?
- error_variance Q_
  Variance of the model error.
- int current_row_
  Index of the row of B currently stored.
- int current_column_
  Index of the column of Q currently stored.
- state_error_variance_row state_error_variance_row_
  Value of the row of B currently stored.
- bool is_adjoint_initialized_
  To indicate if the adjoint variables are allocated or not.
- state_collection q_
  Adjoint variables.
- OutputSaver output_saver_
  Output saver.
- state duplicated_state_
  Duplicated state.
- state state_adjoint_
  Adjoint state.
- state additional_adjoint_term_
  Additional term for adjoint state.

Static Protected Attributes

- static const double Pi_ = 3.141592653589793238462
  PI.

48.5.1 Detailed Description

This class is a clamped-bar model.

Template Parameters

| T          | the type of floating-point numbers. |
48.5.2 Constructor & Destructor Documentation

48.5.2.1 Verdandi::ClampedBar::ClampedBar ( string configuration_file )

Constructor.
It builds allocates the state vectors.

Parameters

| in       | configuration_file | path to the configuration file. |

Definition at line 60 of file ClampedBar.cxx.

48.5.3 Member Function Documentation

48.5.3.1 void Verdandi::ClampedBar::AdjointStateUpdated ( )

Performs some calculations when the update of the adjoint state is done.

Definition at line 950 of file ClampedBar.cxx.

48.5.3.2 double Verdandi::ClampedBar::ApplyOperator ( state & x, bool preserve_state = true, bool update_force = true )

Applies the model to a given state vector.

Parameters

| in,out   | x                  | on entry, the state vector to which the model is applied; on exit, the state vector after the model is applied. |
| in       | preserve_state     | Boolean to indicate if the model state has to be preserved. |

Returns

The time associated with x on exit plus one time step.

Warning

The time of the model has to be preserved.

Definition at line 586 of file ClampedBar.cxx.

48.5.3.3 double Verdandi::ClampedBar::ApplyTangentLinearOperator ( state & increment_state )

Applies the tangent linear model to a given vector.

Parameters

| in,out      | increment | the increment. |

Returns

The time associated with x on exit. This time should be the model time plus one time step.

Definition at line 628 of file ClampedBar.cxx.
48.5.3.4 void Verdandi::ClampedBar::BackwardAdjoint ( state & observation_term )

Performs one step backward in adjoint model.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>observation_term</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$H^T R^{-1} (y - Hx)$.</td>
</tr>
</tbody>
</table>

Definition at line 384 of file ClampedBar.cxx.

48.5.3.5 ClampedBar<T> & Verdandi::ClampedBar::GetAdditionalAdjointTerm ( )

Returns a term that will be added to the adjoint state.

Returns

The additional adjoint term.

Definition at line 972 of file ClampedBar.cxx.

48.5.3.6 double Verdandi::ClampedBar::GetAdditionalCostTerm ( )

Returns a term that will be added to the cost function.

Returns

The additional cost term.

Definition at line 985 of file ClampedBar.cxx.

48.5.3.7 ClampedBar<T> & Verdandi::ClampedBar::GetAdjointState ( )

Returns the adjoint state vector.

Returns

The adjoint state vector.

Definition at line 877 of file ClampedBar.cxx.

48.5.3.8 ClampedBar<T> & error_variance & Verdandi::ClampedBar::GetErrorVariance ( )

Returns the model error variance.

Returns

The model error variance.

Note

If the matrix is empty, it is then assumed there is no model error.

Definition at line 1131 of file ClampedBar.cxx.
ClampedBar\texttt{< T >}::\texttt{state} \& Verdandi::ClampedBar::GetFullState \texttt{()}

Provides the full state vector.

Returns

\texttt{state} the full state vector.

Definition at line 857 of file ClampedBar.cxx.

\hspace{1cm} \begin{tabular}{l}
\texttt{int Verdandi::ClampedBar::GetNstate ( ) const} \\
\texttt{Returns the state vector size.} \\
\texttt{Returns} \\
\texttt{The state vector size.} \\
\end{tabular}

Definition at line 760 of file ClampedBar.cxx.

\hspace{1cm} \begin{tabular}{l}
\texttt{ClampedBar\texttt{< T >}::\texttt{state} \& Verdandi::ClampedBar::GetState \texttt{()}} \\
\texttt{Provides the reduced state vector.} \\
\texttt{Returns} \\
\texttt{state the reduced state vector.} \\
\end{tabular}

Definition at line 808 of file ClampedBar.cxx.

\hspace{1cm} \begin{tabular}{l}
\texttt{void Verdandi::ClampedBar::GetStateCopy ( \texttt{state} \& \texttt{x} )} \\
\texttt{Provides the reduced state vector.} \\
\texttt{The state vector is duplicated.} \\
\texttt{Parameters} \\
\begin{tabular}{|c|c|}
\hline
\texttt{\textbf{out}} & \texttt{\textbf{state}} & the reduced state vector. \\
\hline
\end{tabular} \\
\texttt{Definition at line 773 of file ClampedBar.cxx.} \\
\end{tabular}

\hspace{1cm} \begin{tabular}{l}
\texttt{ClampedBar\texttt{< T >}::\texttt{state\_error\_variance} \& Verdandi::ClampedBar::GetStateErrorVariance \texttt{()}} \\
\texttt{Returns the background error covariance matrix ( B).} \\
\texttt{Returns} \\
\texttt{The matrix of the background error covariance.} \\
\end{tabular}

Definition at line 1017 of file ClampedBar.cxx.

\hspace{1cm} \begin{tabular}{l}
\texttt{const ClampedBar\texttt{< T >}::\texttt{state\_error\_variance} \& Verdandi::ClampedBar::GetStateErrorVariance \texttt{() const}} \\
\texttt{Returns the background error covariance matrix ( B).} \\
\end{tabular}
Returns
The matrix of the background error covariance.

Definition at line 1029 of file ClampedBar.cxx.

48.5.3.15  

**Verdandi::ClampedBar< T >::state_error_variance & Verdandi::ClampedBar::GetStateErrorVarianceInverse ( )**

const

Returns the inverse of the background error variance ($B^{-1}$).

Returns
The inverse of the background error variance ($B^{-1}$).

Definition at line 1117 of file ClampedBar.cxx.

48.5.3.16  

**Verdandi::ClampedBar< T >::state_error_variance & Verdandi::ClampedBar::GetStateErrorVarianceProjector ( )**

Returns the matrix $L$ in the decomposition of the state error covariance matrix ($B$) as a product $LUL^T$.

Returns
The matrix $L$.

Definition at line 1043 of file ClampedBar.cxx.

48.5.3.17  

**Verdandi::ClampedBar< T >::state_error_variance_reduced & Verdandi::ClampedBar::GetStateErrorVarianceReduced ( )**

Returns the matrix $U$ in the decomposition of the state error covariance matrix ($B$) as a product $LUL^T$.

Returns
The matrix $U$.

Definition at line 1084 of file ClampedBar.cxx.

48.5.3.18  

**Verdandi::ClampedBar< T >::state_error_variance_row & Verdandi::ClampedBar::GetStateErrorVarianceRow ( int row )**

Computes a row of the background error covariance matrix $B$.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>row</th>
<th>row index.</th>
</tr>
</thead>
</table>

Returns
The value of row number $row$.

Definition at line 999 of file ClampedBar.cxx.

48.5.3.19  

**Verdandi::ClampedBar< T >::state & Verdandi::ClampedBar::GetStateLowerBound ( )**

Provides the state lower bound.
Returns

The state lower bound (componentwise).

Definition at line 833 of file ClampedBar.cxx.

48.5.3.20 ClampedBar< T >::state & Verdandi::ClampedBar::GetStateUpperBound ( )

Provides the state upper bound.

Returns

The state upper bound (componentwise).

Definition at line 845 of file ClampedBar.cxx.

48.5.3.21 ClampedBar< T >::tangent_linear_operator& Verdandi::ClampedBar::GetTangentLinearOperator ( )

Gets the matrix of the tangent linear model.

Returns

The matrix of the tangent linear model.

Definition at line 721 of file ClampedBar.cxx.

48.5.3.22 double Verdandi::ClampedBar::GetTime ( ) const

Returns the current time.

Returns

The current time.

Definition at line 738 of file ClampedBar.cxx.

48.5.3.23 bool Verdandi::ClampedBar::HasFinished ( ) const

Checks whether the model has finished.

Returns

True if no more data assimilation is required, false otherwise.

Definition at line 348 of file ClampedBar.cxx.

48.5.3.24 void Verdandi::ClampedBar::Initialize ( string configuration_file )

Initializes the model.

Parameters

| in | configuration_file | configuration file. |

Definition at line 92 of file ClampedBar.cxx.
48.5.3.25 void Verdandi::ClampedBar::Save ( )

Saves the simulated data.
It saves the displacement ‘disp_0_’ and the velocity ‘velo_0_’.
Definition at line 372 of file ClampedBar.cxx.

48.5.3.26 void Verdandi::ClampedBar::SetStateCopy ( state & x )

Sets the reduced state vector.
Before setting the reduced state vector, special requirements can be enforced; e.g. positivity requirement or inferior and superior limits.

Parameters

| in | state | the reduced state vector. |

Definition at line 789 of file ClampedBar.cxx.

48.5.3.27 void Verdandi::ClampedBar::SetTime ( double time )

Sets the time of the model to a given time.

Parameters

| in | time | a given time. |

Definition at line 749 of file ClampedBar.cxx.

The documentation for this class was generated from the following files:
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/model/ClampedBar.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/model/ClampedBar.cxx

48.6 ClassTest Class Reference

This class is a test class to run an example.

Public Member Functions

- string GetName () const

  Returns the name of the class.

- void MemberFunction ()

  Calls the logger with an "ok" message.

48.6.1 Detailed Description

This class is a test class to run an example.

48.6.2 Member Function Documentation
48.6.2.1 string ClassTest::GetName ( ) const [inline]

Returns the name of the class.

Returns

The name of the class.

Definition at line 19 of file basic_example_logger.cpp.

The documentation for this class was generated from the following file:

– /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/doc/example/basic_example_logger.cpp

48.7 Verdandi::DiagonalMatrix Class Reference

Diagonal covariance error matrix.

#include <DiagonalMatrix.hxx>

Public Member Functions

– DiagonalMatrix (int dimension, T variance)

Default constructor.

– int GetM () const

Returns the number of rows.

– int GetN () const

Returns the number of columns.

– T operator() (int i, int j) const

Access to one matrix entry.

– void GetRow (int i, Vector<T> &row) const

Access to a row.

– void GetCol (int i, Vector<T> &column) const

Access to a column.

Protected Attributes

– int dimension_

Size of the matrix.

– T variance_

Variance.

48.7.1 Detailed Description

Diagonal covariance error matrix.

This class implements a covariance error matrix that is diagonal.
48.7.2 Constructor & Destructor Documentation

48.7.2.1 Verdandi::DiagonalMatrix::DiagonalMatrix ( int dimension, T variance )

Default constructor.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>dimension</th>
<th>size of the matrix.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>variance</td>
<td>value on the diagonal.</td>
</tr>
</tbody>
</table>

Definition at line 45 of file DiagonalMatrix.cxx.

48.7.3 Member Function Documentation

48.7.3.1 void Verdandi::DiagonalMatrix::GetCol ( int j, Vector< T > & column ) const

Access to a column.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>j</th>
<th>column index.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>column</td>
<td>the column #j.</td>
</tr>
</tbody>
</table>

Definition at line 114 of file DiagonalMatrix.cxx.

48.7.3.2 int Verdandi::DiagonalMatrix::GetM ( ) const [inline]

Returns the number of rows.

Returns

The number of rows.

Definition at line 61 of file DiagonalMatrix.cxx.

48.7.3.3 int Verdandi::DiagonalMatrix::GetN ( ) const [inline]

Returns the number of columns.

Returns

The number of columns.

Definition at line 72 of file DiagonalMatrix.cxx.

48.7.3.4 void Verdandi::DiagonalMatrix::GetRow ( int i, Vector< T > & row ) const

Access to a row.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>i</th>
<th>row index.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>row</td>
<td>the row #i.</td>
</tr>
</tbody>
</table>

Definition at line 100 of file DiagonalMatrix.cxx.
48.7.5 Verdandi::DiagonalMatrix::operator() (int i, int j) const

Access to one matrix entry.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>i</td>
</tr>
<tr>
<td>in</td>
<td>j</td>
</tr>
</tbody>
</table>

Returns

The value of element \((i, j)\).

Definition at line 85 of file DiagonalMatrix.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/error/DiagonalMatrix.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/error/DiagonalMatrix.cxx

48.8 Verdandi::DiscountedRidgeRegression Class Reference

Inheritance diagram for Verdandi::DiscountedRidgeRegression:

```
Verdandi::BaseForecaster<T>
    ↑
Verdandi::DiscountedRidgeRegression
```

Public Member Functions

- **DiscountedRidgeRegression ()**
  
  Default constructor.

- **DiscountedRidgeRegression (T penalization, T gamma, T p)**
  
  Main constructor.

- **virtual ~DiscountedRidgeRegression ()**
  
  Destructor.

- **virtual void SetParameter (const Vector<T> &parameter)**
  
  Sets the vector of parameters.

- **T GetPenalization () const**
  
  Returns the penalization.

- **void SetPenalization (T penalization)**
  
  Sets the penalization.

- **void SetGamma (T gamma)**
  
  Sets the parameter gamma.

- **const Vector<T> & GetInitialWeight () const**
  
  Returns the initial weight vector.

- **void SetInitialWeight (const Vector<T> &initial_weight_)**
  
  Sets the initial weight vector.

- **virtual void Init (const Vector3<T> &ensemble, const Vector2<T> &observation)**
  
  Prepares the aggregation.
virtual void AggregateSpinUpPeriod (int t, const Matrix &simulation_data, const Vector &observation_data, const Vector &ensemble, const Vector &aggregated_simulation)

Aggregates in the spin-up period at a given time.

virtual void ComputeWeight (int t, const Matrix &simulation_data, const Vector &observation_data, const Vector &ensemble, const Vector &observation, const Matrix &weight, Vector &aggregated_simulation)

Computes aggregation weights at a given time.

virtual void ComputeWeightNonSequential (int t, const Vector &ensemble, const Vector &aggregated_simulation)

Computes aggregation weights at a given time.

int GetNspinup () const

void SetNspinup (int Nspinup)

int GetNparameter () const

const Vector &T parameter_

bool IsConvex () const

virtual void Aggregate (const Vector &ensemble, const Vector &aggregated_simulation)

virtual void Aggregate (const Vector &ensemble, const Vector &aggregated_simulation)

virtual void Aggregate (const Vector &ensemble, const Vector &aggregated_simulation)

virtual void Aggregate (const Vector &ensemble, const Vector &aggregated_simulation)

virtual void AggregateThreshold (const Vector &ensemble, const Vector &aggregated_simulation)

virtual void AggregateThreshold (const Vector &ensemble, const Vector &aggregated_simulation)

virtual void AggregateSpinUpPeriodThreshold (int t, const Matrix &simulation_data, const Vector &ensemble, Vector &aggregated_simulation)

void AggregateMean (int t, const Matrix &ensemble, const Vector &aggregated_simulation)

void AggregateZero (int t, const Vector &ensemble, const Vector &aggregated_simulation)

Protected Attributes

T penalization_

Penalization.

T gamma_

Parameter of the sequence of discount factors.

T p_

Parameter of the sequence of discount factors.

Vector &T initial_weight_

Initial weights.

int Nspinup_

Number of step in the spin-up period.

Vector &T parameter_

Parameters.

bool is_convex_

Is the aggregation procedure convex?
48.8.1 Constructor & Destructor Documentation

48.8.1.1 Verdandi::DiscountedRidgeRegression::DiscountedRidgeRegression ( )

Default constructor.
The penalization is set to 0, the parameter gamma is set to 0 and the parameter p is set to 2.
Definition at line 38 of file DiscountedRidgeRegression.cxx.

48.8.1.2 Verdandi::DiscountedRidgeRegression::DiscountedRidgeRegression ( T penalization, T gamma, T p )

Main constructor.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>penalization</th>
<th>penalization.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>gamma</td>
<td>parameter gamma.</td>
</tr>
<tr>
<td>in</td>
<td>p</td>
<td>parameter p.</td>
</tr>
</tbody>
</table>

Definition at line 56 of file DiscountedRidgeRegression.cxx.

48.8.2 Member Function Documentation

48.8.2.1 void Verdandi::DiscountedRidgeRegression::AggregateSpinUpPeriod ( int t, const Matrix< T > & simulation_data, const Vector< T > & observation_data, const Vector3< T > & ensemble, const Vector2< T > & observation, Matrix< T > & weight, Vector2< T > & aggregated ) [virtual]

Aggregates in the spin-up period at a given time.
It carries out the sequential aggregation and returns the aggregated forecast together with the aggregation weights.
This method only carries out the aggregation at one time.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>t</th>
<th>time index at which the aggregation is carried out.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>simulation_data</td>
<td>the ensemble data to be aggregated indexed by member and location.</td>
</tr>
<tr>
<td>in</td>
<td>observation_data</td>
<td>the observational data, indexed by location.</td>
</tr>
<tr>
<td>in</td>
<td>ensemble</td>
<td>the ensemble data indexed by member, time and location.</td>
</tr>
<tr>
<td>in</td>
<td>observation</td>
<td>the observational data, indexed by time and location.</td>
</tr>
<tr>
<td>out</td>
<td>weight</td>
<td>aggregation weights, indexed by member and time.</td>
</tr>
<tr>
<td>out</td>
<td>aggregated</td>
<td>the aggregated output, indexed by time and member.</td>
</tr>
</tbody>
</table>

Reimplemented from Verdandi::BaseForecaster< T >.
Definition at line 192 of file DiscountedRidgeRegression.cxx.

48.8.2.2 void Verdandi::DiscountedRidgeRegression::ComputeWeight ( int t, const Matrix< T > & simulation_data, const Vector< T > & observation_data, const Vector3< T > & ensemble, const Vector2< T > & observation, const Matrix< T > & weight, Vector< T > & weight_vector ) [inline, virtual]

Computes aggregation weights at a given time.
It computes the aggregation weights at one time. This method updates the weights at time t based on past weights.
48.8 Verdandi::DiscountedRidgeRegression Class Reference

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{t}</td>
<td>time index at which the aggregation is carried out.</td>
</tr>
<tr>
<td>\textit{simulation_data}</td>
<td>the ensemble data to be aggregated indexed by member and location.</td>
</tr>
<tr>
<td>\textit{observation_data}</td>
<td>the observational data, indexed by location.</td>
</tr>
<tr>
<td>\textit{ensemble}</td>
<td>the ensemble data indexed by member, time and location.</td>
</tr>
<tr>
<td>\textit{observation}</td>
<td>the observational data, indexed by time and location.</td>
</tr>
<tr>
<td>\textit{weight}</td>
<td>past aggregation weights, indexed by member and time.</td>
</tr>
<tr>
<td>\textit{weight_vector}</td>
<td>aggregation weights for time \textit{t}.</td>
</tr>
</tbody>
</table>

Reimplemented from Verdandi::BaseForecaster< \textit{T} >.

Definition at line 227 of file DiscountedRidgeRegression.hxx.

48.8.2.3 void Verdandi::DiscountedRidgeRegression::ComputeWeightNonSequential ( int \textit{t}, const Vector3< \textit{T} >& \textit{ensemble}, const Vector2< \textit{T} >& \textit{observation}, Vector< \textit{T} >& \textit{weight\_vector} ) [virtual]

Computes aggregation weights at a given time.

It computes the aggregation weights at one time. This method computes the weights at time \textit{t} without knowledge of the past weights.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{t}</td>
<td>time index at which the aggregation is carried out.</td>
</tr>
<tr>
<td>\textit{ensemble}</td>
<td>the ensemble data indexed by member, time and location.</td>
</tr>
<tr>
<td>\textit{observation}</td>
<td>the observational data, indexed by time and location.</td>
</tr>
<tr>
<td>\textit{weight_vector}</td>
<td>aggregation weights for time \textit{t}.</td>
</tr>
</tbody>
</table>

Reimplemented from Verdandi::BaseForecaster< \textit{T} >.

Definition at line 251 of file DiscountedRidgeRegression.hxx.

48.8.2.4 const Vector< \textit{T} >& Verdandi::DiscountedRidgeRegression::GetInitialWeight ( ) const

Returns the initial weight vector.

Returns

The initial weight vector.

Definition at line 140 of file DiscountedRidgeRegression.hxx.

48.8.2.5 \textit{T} Verdandi::DiscountedRidgeRegression::GetPenalization ( ) const

Returns the penalization.

Returns

The penalization.

Definition at line 103 of file DiscountedRidgeRegression.hxx.

48.8.2.6 void Verdandi::DiscountedRidgeRegression::Init ( const Vector3< \textit{T} >& \textit{ensemble}, const Vector2< \textit{T} >& \textit{observation} ) [virtual]

Prepares the aggregation.

This method should be called before an aggregation method is called.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>ensemble</th>
<th>the ensemble data indexed by member, time and location.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>observation</td>
<td>the observational data, indexed by time and location.</td>
</tr>
</tbody>
</table>

Reimplemented from Verdandi::BaseForecaster<T>.
Definition at line 168 of file DiscountedRidgeRegression.cxx.

48.8.2.7 void Verdandi::DiscountedRidgeRegression::SetGamma ( T gamma )

Sets the parameter gamma.

Parameters

| in     | gamma     | the parameter gamma.                                |

Definition at line 128 of file DiscountedRidgeRegression.cxx.

48.8.2.8 void Verdandi::DiscountedRidgeRegression::SetInitialWeight ( const Vector<T> & initial_weight )

Sets the initial weight vector.

Parameters

| in     | initial_weight | the new initial weight vector.                      |

Definition at line 152 of file DiscountedRidgeRegression.cxx.

48.8.2.9 void Verdandi::DiscountedRidgeRegression::SetParameter ( const Vector<T> & parameter ) [virtual]

Sets the vector of parameters.

Parameters

| in     | parameter | new vector of parameters with: the penalization, the parameter gamma and the parameter p. |

Reimplemented from Verdandi::BaseForecaster<T>.
Definition at line 81 of file DiscountedRidgeRegression.cxx.

48.8.2.10 void Verdandi::DiscountedRidgeRegression::SetPenalization ( T penalization )

Sets the penalization.

Parameters

| in     | penalization | penalization.                                      |

Definition at line 115 of file DiscountedRidgeRegression.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/sequential_aggregation/DiscountedRidgeRegression.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/sequential_aggregation/DiscountedRidgeRegression.cxx
48.9 Verdandi::EnsembleKalmanFilter Class Reference

This class implements the ensemble Kalman filter.

```cpp
#include <EnsembleKalmanFilter.hxx>
```

Inheritance diagram for Verdandi::EnsembleKalmanFilter:

```
Verdandi::VerdandiBase
    └── Verdandi::EnsembleKalmanFilter
```

Public Types

- `typedef Model::state model_state`
  
  Type of the model state vector.

- `typedef ObservationManager::observation observation observation`
  
  Type of the observation vector.

- `typedef Model::state::value_type Ts`
  
  Value type of the model state vector.

- `typedef ObservationManager::observation::value_type To`
  
  Value type of the observations.

- `typedef Model::uncertain_parameter uncertain_parameter`
  
  Type of an uncertain parameter.

- `typedef Model::uncertain_parameter::value_type Tp`
  
  Value type of an uncertain parameter.

- `typedef vector<model_state> ensemble`
  
  Type of the ensemble of state vectors.

Public Member Functions

- `EnsembleKalmanFilter ()`
  
  Main constructor.

- `~EnsembleKalmanFilter ()`
  
  Destructor.

- `void Initialize (string configuration_file, bool initialize_model=true, bool initialize_observation_manager=true, bool initialize_perturbation_manager=true)`
  
  Initializes the ensemble Kalman filter.

- `void Initialize (VerdandiOps &configuration, bool initialize_model=true, bool initialize_observation_manager=true, bool initialize_perturbation_manager=true)`
  
  Initializes the ensemble Kalman filter.

- `void InitializeEnsemble ()`
  
  Initialization of the perturbations.

- `void InitializeStep ()`
  
  Initializes a step for the method.

- `void Forward ()`
  
  Performs a step forward for the ensemble.

- `void Prediction ()`
  
  Performs a forecast step for the ensemble.

- `void Analyze ()`
  
  Performs an analysis step for the ensemble.
Computes an analysis.

- `void FinalizeStep ()`
  Finalizes a step for the model.
- `void Finalize ()`
  Finalizes the model.
- `bool HasFinished () const`
  Checks whether the model has finished.
- `Model & GetModel ()`
  Returns the model.
- `ObservationManager & GetObservationManager ()`
  Returns the observation manager.
- `PerturbationManager & GetPerturbationManager ()`
  Returns the perturbation manager.
- `OutputSaver & GetOutputSaver ()`
  Returns the output saver.
- `string GetName () const`
  Returns the name of the class.
- `void Message (string message)`
  Receives and handles a message.

Static Public Member Functions

- `static void StaticMessage (void *object, string message)`
  Receives and handles a message with a static method.

Protected Member Functions

- `template<class T0 , class Allocator0 >
  void Fill (Vector<T0, Collection, Allocator0> &in, string pdf)`
  Fills an input vector collection according to its probability distribution.
- `template<class T0 , class Storage0 , class Allocator0 >
  void Fill (Vector<T0, Storage0, Allocator0> &in, string pdf)`
  Fills an input vector according to its probability distribution.
- `template<class T0 , class Storage0 , class Allocator0 >
  void SetDimension (Vector<T0, Storage0, Allocator0> &in, Vector<T0, Storage0, Allocator0> &out)`
  Allocates an output vector to the dimension of the input vector.
- `template<class T0 , class Allocator0 >
  void SetDimension (Vector<T0, Collection, Allocator0> &in, Vector<T0, Collection, Allocator0> &out)`
  Allocates an output vector collection to the dimension of the input vector collection.

Protected Attributes

- `Model model_`
  Underlying model.
- `ObservationManager observation_manager_`
  Observation manager.
- `PerturbationManager perturbation_manager_`
  Perturbation managers.
- `int iteration_`
  Iteration.
48.9.1 Detailed Description

This class implements the ensemble Kalman filter.

48.9.2 Constructor & Destructor Documentation

48.9.2.1 Verdandi::EnsembleKalmanFilter::EnsembleKalmanFilter ( )

Main constructor.
Builds the driver.
Definition at line 42 of file EnsembleKalmanFilter.cxx.
48.9.3 Member Function Documentation

48.9.3.1 template<class T0 , class Allocator0 > void Verdandi::EnsembleKalmanFilter::Fill ( Vector<T0, Collection, Allocator0> & in, string pdf ) [protected]

Fills an input vector collection according to its probability distribution.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in, out</td>
<td>in</td>
</tr>
<tr>
<td>in</td>
<td>pdf</td>
</tr>
</tbody>
</table>

Definition at line 915 of file EnsembleKalmanFilter.cxx.

48.9.3.2 template<class T0 , class Storage0 , class Allocator0 > void Verdandi::EnsembleKalmanFilter::Fill ( Vector<T0, Storage0, Allocator0> & in, string pdf ) [protected]

Fills an input vector according to its probability distribution.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in, out</td>
<td>in</td>
</tr>
<tr>
<td>in</td>
<td>pdf</td>
</tr>
</tbody>
</table>

Definition at line 939 of file EnsembleKalmanFilter.cxx.

48.9.3.3 Model & Verdandi::EnsembleKalmanFilter::GetModel ( )

Returns the model.

Returns

The model.

Definition at line 741 of file EnsembleKalmanFilter.cxx.

48.9.3.4 string Verdandi::EnsembleKalmanFilter::GetName ( ) const [virtual]

Returns the name of the class.

Returns

The name of the class.

Reimplemented from Verdandi::VerdandiBase.

Definition at line 795 of file EnsembleKalmanFilter.cxx.

48.9.3.5 ObservationManager & Verdandi::EnsembleKalmanFilter::GetObservationManager ( )

Returns the observation manager.

Returns

The observation manager.

Definition at line 755 of file EnsembleKalmanFilter.cxx.
48.9.3.6 OutputSaver & Verdandi::EnsembleKalmanFilter::GetOutputSaver ( )

Returns the output saver.

Returns
   The output saver.

Definition at line 782 of file EnsembleKalmanFilter.cxx.

48.9.3.7 PerturbationManager & Verdandi::EnsembleKalmanFilter::GetPerturbationManager ( )

Returns the perturbation manager.

Returns
   The perturbation manager.

Definition at line 769 of file EnsembleKalmanFilter.cxx.

48.9.3.8 bool Verdandi::EnsembleKalmanFilter::HasFinished ( ) const

Checks whether the model has finished.

Returns
   True if no more data assimilation is required, false otherwise.

Definition at line 726 of file EnsembleKalmanFilter.cxx.

48.9.3.9 void Verdandi::EnsembleKalmanFilter::Initialize ( string configuration_file, bool initialize_model = true, bool initialize_observation_manager = true, bool initialize_perturbation_manager = true )

Initializes the ensemble Kalman filter.

It reads the configuration and initializes the model and the observation manager. It can also compute an analysis with the model's initial condition.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>configuration_file</th>
<th>configuration file for the method.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>initialize_model</td>
<td>should the model be initialized with a call to Model::Initialize(string)?</td>
</tr>
<tr>
<td>in</td>
<td>initialize_</td>
<td>should the observation manager be initialized with a call to Observation-</td>
</tr>
<tr>
<td></td>
<td>observation_</td>
<td>Manager::Initialize(Model&amp;, string)?</td>
</tr>
<tr>
<td></td>
<td>manager</td>
<td></td>
</tr>
<tr>
<td>in</td>
<td>initialize_</td>
<td>should the perturbation manager be initialized with a call to Perturbation-</td>
</tr>
<tr>
<td></td>
<td>perturbation_</td>
<td>Manager::Initialize(string)?</td>
</tr>
<tr>
<td></td>
<td>manager</td>
<td></td>
</tr>
</tbody>
</table>

Warning
   If initialize_model is set to false, the model should be initialized before calling this function.

Definition at line 105 of file EnsembleKalmanFilter.cxx.
void Verdandi::EnsembleKalmanFilter::Initialize ( VerdandiOps & configuration, bool initialize_model = true, bool initialize_observation_manager = true, bool initialize_perturbation_manager = true )

Initializes the ensemble Kalman filter.

It reads the configuration and initializes the model and the observation manager. It can also compute an analysis with the model's initial condition.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>configuration</th>
<th>configuration for the method.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>initialize_model</td>
<td>should the model be initialized with a call to Model::Initialize(string)?</td>
</tr>
<tr>
<td>in</td>
<td>initialize_observation_manager</td>
<td>should the observation manager be initialized with a call to ObservationManager::Initialize(Model&amp;, string)?</td>
</tr>
<tr>
<td>in</td>
<td>initialize_perturbation_manager</td>
<td>should the perturbation manager be initialized with a call to PerturbationManager::Initialize(string)?</td>
</tr>
</tbody>
</table>

Warning

If initialize_model is set to false, the model should be initialized before calling this function.

Definition at line 136 of file EnsembleKalmanFilter.cxx.

void Verdandi::EnsembleKalmanFilter::InitializeEnsemble ( )

Initialization of the perturbations.

The perturbations of the parameters are generated independently for each member.

Definition at line 311 of file EnsembleKalmanFilter.cxx.

template < class T0 , class Storage0 , class Allocator0 > void Verdandi::EnsembleKalmanFilter::SetDimension ( Vector< T0, Storage0, Allocator0 > & in, Vector< T0, Storage0, Allocator0 > & out ) [protected]

Allocates an output vector to the dimension of the input vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>in</th>
<th>input vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>out</td>
<td>output vector.</td>
</tr>
</tbody>
</table>

Definition at line 958 of file EnsembleKalmanFilter.cxx.

template < class T0 , class Allocator0 > void Verdandi::EnsembleKalmanFilter::SetDimension ( Vector< T0, Collection, Allocator0 > & in, Vector< T0, Collection, Allocator0 > & out ) [protected]

Allocates an output vector collection to the dimension of the input vector collection.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>in</th>
<th>input collection vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>out</td>
<td>output collection vector.</td>
</tr>
</tbody>
</table>

Definition at line 976 of file EnsembleKalmanFilter.cxx.

The documentation for this class was generated from the following files:
This class serves for exceptions raised when an error is found.

#include <Error.hxx>

Inheritance diagram for Verdandi::Error:

```
Verdandi::Error
   `-- Verdandi::ErrorArgument
   `-- Verdandi::ErrorConfiguration
   `-- Verdandi::ErrorIO
   `-- Verdandi::ErrorProcessing
   `-- Verdandi::ErrorPythonUndefined
   `-- Verdandi::ErrorUndefined
```

Public Member Functions

- **Error** (string function="", string comment="")
  *Main constructor.*

- **Error** (string description, string function, string comment)
  *Alternative constructor.*

- virtual ~**Error** ()
  *Destructor.*

- virtual string GetName () const
  *Returns the name of the class.*

- virtual string What ()
  *Delivers information about the error.*

- void CoutWhat ()
  *Delivers information about the error.*

Protected Attributes

- string **description**_
  *Message describing the exception type.*

- string **function**_
  *Name of the function in which the error occurred.*

- string **comment**_
  *A comment about the error.*

48.10.1 Detailed Description

This class serves for exceptions raised when an error is found.

All **Verdandi** exceptions are supposed to inherit from this class.

48.10.2 Constructor & Destructor Documentation

48.10.2.1 Verdandi::Error::Error ( string function = " ", string comment = " ")

*Main constructor.*

**Error** associated with both a function and a comment.
Class Documentation

Parameters

| in | function | function in which the error occurred. |
| in | comment   | comment associated with the error.    |

Definition at line 49 of file Error.cxx.

48.10.2.2 Verdandi::Error::Error ( string description, string function, string comment )

Alternative constructor.

Error associated with a description, a function and a comment.

Parameters

| in | description | short description of the error. |
| in | function    | function in which the error occurred. |
| in | comment     | comment associated with the error. |

Definition at line 70 of file Error.cxx.

48.10.2.3 Verdandi::Error::~Error ( ) [virtual]

Destructor.

Note

Empty.

Definition at line 85 of file Error.cxx.

48.10.3 Member Function Documentation

48.10.3.1 void Verdandi::Error::CoutWhat ( )

Delivers information about the error.

Displays available information, i.e. the error description, the function and/or the comment.

Definition at line 125 of file Error.cxx.

48.10.3.2 string Verdandi::Error::GetName ( ) const [virtual]

Returns the name of the class.

Returns

The name of the class.

Reimplemented in Verdandi::ErrorPythonUndefined, Verdandi::ErrorArgument, Verdandi::ErrorUndefined, Verdandi::ErrorProcessing, Verdandi::ErrorIO, and Verdandi::ErrorConfiguration.

Definition at line 99 of file Error.cxx.

48.10.3.3 string Verdandi::Error::What ( ) [virtual]

Delivers information about the error.
48.11 Verdandi::ErrorArgument Class Reference

This class serves for exceptions raised when a function or a method is called with an erroneous argument.

```cpp
#include <Error.hxx>
```

Inheritance diagram for Verdandi::ErrorArgument:

```
Verdandi::ErrorArgument
        ^
         |   
Verdandi::Error
```

Public Member Functions

- `ErrorArgument (string function="", string comment="")`
  
  Main constructor.
- `string GetName () const`
  
  Returns the name of the class.
- `virtual string What ()`
  
  Delivers information about the error.
- `void CoutWhat ()`
  
  Delivers information about the error.

Protected Attributes

- `string description_`
  
  Message describing the exception type.
- `string function_`
  
  Name of the function in which the error occurred.
- `string comment_`
  
  A comment about the error.

48.11.1 Detailed Description

This class serves for exceptions raised when a function or a method is called with an erroneous argument.
48.11.2 Constructor & Destructor Documentation

48.11.2.1 Verdandi::ErrorArgument::ErrorArgument ( string function = " ", string comment = " " )

Main constructor.

Error associated with both a function and a comment.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>function</th>
<th>function in which the error occurred.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>comment</td>
<td>comment associated with the error.</td>
</tr>
</tbody>
</table>

Definition at line 294 of file Error.cxx.

48.11.3 Member Function Documentation

48.11.3.1 void Verdandi::Error::CoutWhat ( ) [inherited]

Delivers information about the error.
Displays available information, i.e. the error description, the function and/or the comment.
Definition at line 125 of file Error.cxx.

48.11.3.2 string Verdandi::ErrorArgument::GetName ( ) const [virtual]

Returns the name of the class.

Returns

The name of the class.

Reimplemented from Verdandi::Error.
Definition at line 312 of file Error.cxx.

48.11.3.3 string Verdandi::Error::What ( ) [virtual, inherited]

Delivers information about the error.
Displays available information, i.e. the error description, the function and/or the comment.
Reimplemented in Verdandi::ErrorPythonUndefined, and Verdandi::ErrorUndefined.
Definition at line 109 of file Error.cxx.

The documentation for this class was generated from the following files:

   – /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/share/Error.hxx
   – /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/share/Error.cxx

48.12 Verdandi::ErrorConfiguration Class Reference

This class serves for exceptions raised when an error is found in a configuration.

#include <Error.hxx>

Inheritance diagram for Verdandi::ErrorConfiguration:
Public Member Functions

- **ErrorConfiguration** (string function="", string comment="")
  Main constructor.
- string GetName () const
  Returns the name of the class.
- virtual string What ()
  Delivers information about the error.
- void CoutWhat ()
  Delivers information about the error.

Protected Attributes

- string description_
  Message describing the exception type.
- string function_
  Name of the function in which the error occurred.
- string comment_
  A comment about the error.

48.12.1 Detailed Description

This class serves for exceptions raised when an error is found in a configuration.

48.12.2 Constructor & Destructor Documentation

48.12.2.1 Verdandi::ErrorConfiguration::ErrorConfiguration ( string function = "", string comment = "" )

Main constructor.

**Error** associated with both a function and a comment.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>function</th>
<th>function in which the error occurred.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>comment</td>
<td>comment associated with the error.</td>
</tr>
</tbody>
</table>

Definition at line 141 of file Error.cxx.

48.12.3 Member Function Documentation

48.12.3.1 void Verdandi::Error::CoutWhat ( ) [inherited]

Delivers information about the error.

Displays available information, i.e. the error description, the function and/or the comment.
48.12.3.2 string Verdandi::ErrorConfiguration::GetName ( ) const  [virtual]

Returns the name of the class.

Returns
   The name of the class.

Reimplemented from Verdandi::Error.
Definition at line 159 of file Error.cxx.

48.12.3.3 string Verdandi::Error::What ( )  [virtual, inherited]

Delivers information about the error.
Displays available information, i.e. the error description, the function and/or the comment.
Reimplemented in Verdandi::ErrorPythonUndefined, and Verdandi::ErrorUndefined.
Definition at line 109 of file Error.cxx.

The documentation for this class was generated from the following files:
   – /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/share/Error.hxx
   – /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/share/Error.cxx

48.13 Verdandi::ErrorIO Class Reference

This class serves for exceptions raised when an input/output error occurs.
#include <Error.hxx>

Inheritance diagram for Verdandi::ErrorIO:

```
Verdandi::Error

Verdandi::ErrorIO
```

Public Member Functions

- **ErrorIO** (string function="", string comment="")
  *Main constructor.*
- string **GetName** ( ) const
  *Returns the name of the class.*
- virtual string **What** ( )
  *Delivers information about the error.*
- void **CoutWhat** ( )
  *Delivers information about the error.*
Protected Attributes

- string description_
  
  Message describing the exception type.
- string function_
  
  Name of the function in which the error occurred.
- string comment_
  
  A comment about the error.

48.13.1 Detailed Description

This class serves for exceptions raised when an input/output error occurs.

48.13.2 Constructor & Destructor Documentation

48.13.2.1 Verdandi::ErrorIO::ErrorIO ( string function = "", string comment = "" )

Main constructor.

Error associated with both a function and a comment.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>function</th>
<th>function in which the error occurred.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>comment</td>
<td>comment associated with the error.</td>
</tr>
</tbody>
</table>

Definition at line 175 of file Error.cxx.

48.13.3 Member Function Documentation

48.13.3.1 void Verdandi::Error::CoutWhat ( ) [inherited]

Delivers information about the error.

Displays available information, i.e. the error description, the function and/or the comment.

Definition at line 125 of file Error.cxx.

48.13.3.2 string Verdandi::ErrorIO::GetName ( ) const [virtual]

Returns the name of the class.

Returns

The name of the class.

Reimplemented from Verdandi::Error.

Definition at line 193 of file Error.cxx.

48.13.3.3 string Verdandi::Error::What ( ) [virtual, inherited]

Delivers information about the error.

Displays available information, i.e. the error description, the function and/or the comment.

Reimplemented in Verdandi::ErrorPythonUndefined, and Verdandi::ErrorUndefined.
Definition at line 109 of file Error.cxx.
The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/share/Error.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/share/Error.cxx

### 48.14 Verdandi::ErrorProcessing Class Reference

This class serves for exceptions raised when an error occurs during some data processing.

#include <Error.hxx>

Inheritance diagram for Verdandi::ErrorProcessing:

![Inheritance Diagram](diagram)

#### Public Member Functions

- **ErrorProcessing** (string function="", string comment="")
  
  Main constructor.

- string **GetName** () const
  
  Returns the name of the class.

- virtual string **What** ()
  
  Delivers information about the error.

- void **CoutWhat** ()
  
  Delivers information about the error.

#### Protected Attributes

- string **description_**
  
  Message describing the exception type.

- string **function_**
  
  Name of the function in which the error occurred.

- string **comment_**
  
  A comment about the error.

#### 48.14.1 Detailed Description

This class serves for exceptions raised when an error occurs during some data processing.

#### 48.14.2 Constructor & Destructor Documentation

48.14.2.1 Verdandi::ErrorProcessing::ErrorProcessing ( string function = "", string comment = "" )

Main constructor.

_Error_ associated with both a function and a comment.
### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>function</th>
<th>function in which the error occurred.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>comment</td>
<td>comment associated with the error.</td>
</tr>
</tbody>
</table>

Definition at line 210 of file Error.cxx.

### 48.14.3 Member Function Documentation

#### 48.14.3.1 void Verdandi::Error::CoutWhat ( ) [inherited]

Delivers information about the error.
Displays available information, i.e. the error description, the function and/or the comment.
Definition at line 125 of file Error.cxx.

#### 48.14.3.2 string Verdandi::ErrorProcessing::GetName ( ) const [virtual]

Returns the name of the class.

Returns

The name of the class.

Reimplemented from Verdandi::Error.
Definition at line 228 of file Error.cxx.

#### 48.14.3.3 string Verdandi::Error::What ( ) [virtual, inherited]

Delivers information about the error.
Displays available information, i.e. the error description, the function and/or the comment.
Reimplemented in Verdandi::ErrorPythonUndefined, and Verdandi::ErrorUndefined.
Definition at line 109 of file Error.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/share/Error.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/share/Error.cxx

### 48.15 Verdandi::ErrorPythonUndefined Class Reference

This class serves for exceptions raised when an undefined function or method in a Python module is called.

#include <Error.hxx>

Inheritance diagram for Verdandi::ErrorPythonUndefined:

```
Verdandi::ErrorPythonUndefined
    `-- Verdandi::Error
```

Generated on Mon Sep 28 2015 16:49:58 for Verdandi by Doxygen
Public Member Functions

- `ErrorPythonUndefined` (string function="", string function_name="", string arguments="", string module="", string comment="")
  Main constructor.
- `~ErrorPythonUndefined` ()
  Destructor.
- virtual string `What` ()
  Delivers information about the error.
- string `GetName` () const
  Returns the name of the class.
- void `CoutWhat` ()
  Delivers information about the error.

Protected Attributes

- string `function_name_`
  Name of the Python function.
- string `arguments_`
  List of arguments of the function.
- string `module_`
  Name of the Python module.
- string `description_`
  Message describing the exception type.
- string `function_`
  Name of the function in which the error occurred.
- string `comment_`
  A comment about the error.

48.15.1 Detailed Description

This class serves for exceptions raised when an undefined function or method in a Python module is called.

48.15.2 Constructor & Destructor Documentation

48.15.2.1 Verdandi::ErrorPythonUndefined::ErrorPythonUndefined ( string function = "", string function_name = "", string arguments = "", string module = "", string comment = ""

Main constructor.

Error associated with both a function and a comment.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>function</th>
<th>function in which the error occurred.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>function_name</td>
<td>name of the python function called.</td>
</tr>
<tr>
<td>in</td>
<td>arguments</td>
<td>arguments that must be passed to the function.</td>
</tr>
<tr>
<td>in</td>
<td>module</td>
<td>Python module where the function is searched.</td>
</tr>
<tr>
<td>in</td>
<td>comment</td>
<td>comment associated with the error.</td>
</tr>
</tbody>
</table>

Definition at line 331 of file Error.cxx.
48.15.2 Verdandi::ErrorUndefined Class Reference

48.15.2.2 Verdandi::ErrorPythonUndefined::~ErrorPythonUndefined()

Destructor.

Note

Empty.

Definition at line 357 of file Error.cxx.

48.15.3 Member Function Documentation

48.15.3.1 void Verdandi::Error::CoutWhat()

[inherited]

Delivers information about the error.
Displays available information, i.e. the error description, the function and/or the comment.
Definition at line 125 of file Error.cxx.

48.15.3.2 string Verdandi::ErrorPythonUndefined::GetName() const

[virtual]

Returns the name of the class.
Returns

The name of the class.
Reimplemented from Verdandi::Error.
Definition at line 385 of file Error.cxx.

48.15.3.3 string Verdandi::ErrorPythonUndefined::What()

[virtual]

Delivers information about the error.
Displays available information, i.e. the error description, the function and/or the comment.
Reimplemented from Verdandi::Error.
Definition at line 365 of file Error.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/share/Error.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/share/Error.cxx

48.16 Verdandi::ErrorUndefined Class Reference

This class serves for exceptions raised when an undefined function or method is called.

#include <Error.hxx>

Inheritance diagram for Verdandi::ErrorUndefined:

[Diagram]

Verdandi::Error

Verdandi::ErrorUndefined
Public Member Functions

- **ErrorUndefined** (string function="", string comment="")
  
  Main constructor.
- **virtual string What ()**
  
  Delivers information about the error.
- **string GetName () const**
  
  Returns the name of the class.
- **void CoutWhat ()**
  
  Delivers information about the error.

Protected Attributes

- **string description_**
  
  Message describing the exception type.
- **string function_**
  
  Name of the function in which the error occurred.
- **string comment_**
  
  A comment about the error.

48.16.1 Detailed Description

This class serves for exceptions raised when an undefined function or method is called.

48.16.2 Constructor & Destructor Documentation

48.16.2.1 Verdandi::ErrorUndefined::ErrorUndefined ( string function = "", string comment = "" )

Main constructor.

**Error** associated with both a function and a comment.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>function</th>
<th>function in which the error occurred.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>comment</td>
<td>comment associated with the error.</td>
</tr>
</tbody>
</table>

Definition at line 244 of file Error.cxx.

48.16.3 Member Function Documentation

48.16.3.1 void Verdandi::Error::CoutWhat ( ) [inherited]

Delivers information about the error.

Displays available information, i.e. the error description, the function and/or the comment.

Definition at line 125 of file Error.cxx.

48.16.3.2 string Verdandi::ErrorUndefined::GetName ( ) const [virtual]

Returns the name of the class.
Returns

The name of the class.

Reimplemented from Verdandi::Error.

Definition at line 278 of file Error.cxx.

48.16.3.3 string Verdandi::ErrorUndefined::What ( ) [virtual]

Delivers information about the error.
Displays available information, i.e. the error description, the function and/or the comment.
Reimplemented from Verdandi::Error.

Definition at line 262 of file Error.cxx.

The documentation for this class was generated from the following files:

– /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/share/Error.hxx
– /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/share/Error.cxx

48.17 Verdandi::ExtendedKalmanFilter Class Reference

This class implements the extended Kalman filter.

#include <ExtendedKalmanFilter.hxx>

Inheritance diagram for Verdandi::ExtendedKalmanFilter:

```
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Verdandi::ExtendedKalmanFilter</td>
</tr>
<tr>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>Verdiandi::VerdandiBase</td>
</tr>
<tr>
<td>---------------------------------------------</td>
</tr>
</tbody>
</table>
```

Public Types

- typedef Model::state::value_type Ts
  Value type of the model state.

- typedef Model::state_error_variance_row model_state_error_variance_row
  Type of a row of the background error variance.

- typedef Model::state model_state
  Type of the model state vector.

- typedef Model::matrix_state_observation matrix_state_observation
  Type of the model/observation crossed matrix.

- typedef Model::state_error_variance model_state_error_variance
  Type of the background error variance.

- typedef Model::tangent_linear_operator model_tangent_linear_operator
  Type of the tangent linear model.

- typedef ObservationManager::tangent_linear_operator observation_tangent_linear_operator
  Type of the tangent linear observation operator.

- typedef ObservationManager::tangent_linear_operator_row observation_tangent_linear_operator_row
  Type of a row of the tangent linear observation operator.

- typedef ObservationManager::observation observation
  Type of the observation vector.
Public Member Functions

– ExtendedKalmanFilter ()
  Main constructor.
– ~ExtendedKalmanFilter ()
  Destructor.
– void Initialize (string configuration_file, bool initialize_model=true, bool initialize_observation_manager=true)
  Initializes the extended Kalman filter driver.
– void Initialize (VerdandiOps &configuration, bool initialize_model=true, bool initialize_observation_manager=true)
  Initializes the extended Kalman filter driver.
– void InitializeStep ()
  Initializes a step for the model.
– void Prediction ()
  Performs a forecast step.
– void Forward ()
  Performs a step forward, with extended Kalman filter.
– void Analyze ()
  Computes an analysis.
– void FinalizeStep ()
  Finalizes a step for the model.
– void Finalize ()
  Finalizes the model.
– void PropagateCovarianceMatrix ()
  Computes Covariance.
– void PropagateCovarianceMatrix_vector ()
  Computes covariance.
– void PropagateCovarianceMatrix_matrix ()
  Computes covariance.
– void ComputeBLUE (const observation &innovation, model_state &state)
  Computes BLUE for Extended Kalman Filter.
– bool HasFinished ()
  Checks whether the model has finished.
– Model & GetModel ()
  Returns the model.
– ObservationManager & GetObservationManager ()
  Returns the observation manager.
– OutputSaver & GetOutputSaver ()
  Returns the output saver.
– string GetName () const
  Returns the name of the class.
– void Message (string message)
  Receives and handles a message.

Static Public Member Functions

– static void StaticMessage (void ∗object, string message)
  Receives and handles a message with a static method.
Protected Attributes

- Model model_
  Underlying model.
- ObservationManager observation_manager_
  Observation manager.
- model_state_error_variance state_error_variance_
  Background error covariance matrix (B).
- int iteration_
  Iteration.
- string configuration_file_
  Path to the configuration file.
- string model_configuration_file_
  Path to the model configuration file.
- string observation_configuration_file_
  Path to the configuration file for the observation manager.
- map<string, bool> option_display_
  Display options.
- int Nstate_
  Dimension of the state.
- int Nobservation_
  Number of observations.
- bool analyze_first_step_
  Should an analysis be computed at the first step?
- double time_
  Current time.
- string blue_computation_
  Computation mode for BLUE: "vector" or "matrix".
- string covariance_computation_
  Computation mode for covariance: "vector" or "matrix".
- OutputSaver output_saver_
  Output saver.

48.17.1 Detailed Description

This class implements the extended Kalman filter.

48.17.2 Constructor & Destructor Documentation

48.17.2.1 Verdandi::ExtendedKalmanFilter::ExtendedKalmanFilter ( )

Main constructor.
Builds the driver.
Definition at line 44 of file ExtendedKalmanFilter.cxx.
48.17.3 Member Function Documentation

48.17.3.1 void Verdandi::ExtendedKalmanFilter::Analyze ( )

Computes an analysis. Whenever observations are available, it computes BLUE.
Definition at line 262 of file ExtendedKalmanFilter.cxx.

48.17.3.2 void Verdandi::ExtendedKalmanFilter::ComputeBLUE ( const observation & innovation, model_state & state )

Computes BLUE for Extended Kalman Filter. The state is updated by the combination of background state and innovation. It computes the BLUE (best linear unbiased estimator).

Parameters

| In | state_vector | the state vector to analyze. |

Definition at line 387 of file ExtendedKalmanFilter.cxx.

48.17.3.3 Model & Verdandi::ExtendedKalmanFilter::GetModel ( )

Returns the model.

Returns

The model.

Definition at line 431 of file ExtendedKalmanFilter.cxx.

48.17.3.4 string Verdandi::ExtendedKalmanFilter::GetName ( ) const [virtual]

Returns the name of the class.

Returns

The name of the class.

Reimplemented from Verdandi::VerdandiBase.
Definition at line 467 of file ExtendedKalmanFilter.cxx.

48.17.3.5 ObservationManager & Verdandi::ExtendedKalmanFilter::GetObservationManager ( )

Returns the observation manager.

Returns

The observation manager.

Definition at line 444 of file ExtendedKalmanFilter.cxx.

48.17.3.6 OutputSaver & Verdandi::ExtendedKalmanFilter::GetOutputSaver ( )

Returns the output saver.
Returns

The output saver.

Definition at line 456 of file ExtendedKalmanFilter.hxx.

48.17.3.7 bool Verdandi::ExtendedKalmanFilter::HasFinished ( )

Checks whether the model has finished.

Returns

True if no more data assimilation is required, false otherwise.

Definition at line 407 of file ExtendedKalmanFilter.hxx.

48.17.3.8 void Verdandi::ExtendedKalmanFilter::Initialize ( string configuration, bool initialize_model = true, bool initialize_observation_manager = true )

Initializes the extended Kalman filter driver.

Initializes the model and the observation manager.

Definition at line 76 of file ExtendedKalmanFilter.hxx.

48.17.3.9 void Verdandi::ExtendedKalmanFilter::Initialize ( VerdandiOps & configuration, bool initialize_model = true, bool initialize_observation_manager = true )

Initializes the extended Kalman filter driver.

Initializes the model and the observation manager. Optionally computes the analysis of the first step.

Definition at line 90 of file ExtendedKalmanFilter.hxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/ExtendedKalmanFilter.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/ExtendedKalmanFilter.hxx

48.18 Verdandi::ForwardDriver Class Reference

This class simply performs a forward simulation.

#include <ForwardDriver.hxx>

Inheritance diagram for Verdandi::ForwardDriver:

```
Verdandi::VerdandiBase

Verdandi::ForwardDriver
```

Public Types

- typedef Model::state model_state
  
  Type of the model state vector.
Public Member Functions

- **ForwardDriver ()**
  Main constructor.

- **∼ForwardDriver ()**
  Destructor.

- **void Initialize (string configuration_file, bool initialize_model=true)**
  Initializes the simulation.

- **void Initialize (VerdandiOps &configuration, bool initialize_model=true)**
  Initializes the simulation.

- **void InitializeStep ()**
  Initializes the model before a time step.

- **void Forward ()**
  Performs a step forward without optimal interpolation.

- **void FinalizeStep ()**
  Finalizes a step for the model.

- **void Finalize ()**
  Finalizes the model.

- **bool HasFinished ()**
  Checks whether the model has finished.

- **Model & GetModel ()**
  Returns the model.

- **OutputSaver & GetOutputSaver ()**
  Returns the output saver.

- **string GetName () const**
  Returns the name of the class.

- **void Message (string message)**
  Receives and handles a message.

Static Public Member Functions

- **static void StaticMessage (void *object, string message)**
  Receives and handles a message with a static method.

Protected Attributes

- **Model model_**
  Underlying model.

- **int iteration_**
  Iteration.

- **Vector<double> time_**
  Time vector.

- **string configuration_file_**
  Path to the configuration file.

- **string model_configuration_file_**
  Path to the model configuration file.

- **map<string, bool> option_display_**
  Display options.

- **OutputSaver output_saver_**
  Output saver.
48.18.1 Detailed Description

This class simply performs a forward simulation.

48.18.2 Constructor & Destructor Documentation

48.18.2.1 Verdandi::ForwardDriver::ForwardDriver ( )

Main constructor.
Builds the driver and reads option keys in the configuration file.

Parameters

| in  | configuration | configuration file. |

Definition at line 43 of file ForwardDriver.cxx.

48.18.3 Member Function Documentation

48.18.3.1 Model & Verdandi::ForwardDriver::GetModel ( )

Returns the model.

Returns

The model.

Definition at line 323 of file ForwardDriver.cxx.

48.18.3.2 string Verdandi::ForwardDriver::GetName ( ) const [virtual]

Returns the name of the class.

Returns

The name of the class.

Reimplemented from Verdandi::VerdandiBase.
Definition at line 346 of file ForwardDriver.cxx.

48.18.3.3 OutputSaver & Verdandi::ForwardDriver::GetOutputSaver ( )

Returns the output saver.

Returns

The output saver.

Definition at line 335 of file ForwardDriver.cxx.

48.18.3.4 bool Verdandi::ForwardDriver::HasFinished ( )

Checks whether the model has finished.
Returns

True if no more data assimilation is required, false otherwise.

Definition at line 312 of file ForwardDriver.cxx.

48.18.3.5 void Verdandi::ForwardDriver::Initialize ( string configuration_file, bool initialize_model = true )

Initializes the simulation.
Initializes the model.

Parameters

| in | configuration_file | configuration file to be given to the model initialization method. |

Definition at line 88 of file ForwardDriver.cxx.

48.18.3.6 void Verdandi::ForwardDriver::Initialize ( VerdandiOps & configuration, bool initialize_model = true )

Initializes the simulation.
Initializes the model.

Parameters

| in | configuration | configuration file to be given to the model initialization method. |

Definition at line 102 of file ForwardDriver.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/ForwardDriver.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/ForwardDriver.cxx

48.19 Verdandi::FourDimensionalVariational Class Reference

This class implements 4D-Var.

#include <FourDimensionalVariational.hxx>

Inheritance diagram for Verdandi::FourDimensionalVariational:

```
Verdandi::VerdandiBase
   ↓
Verdandi::FourDimensionalVariational
```

Public Types

- typedef Model::state::value_type Ts
  
  Value type of the model state.

- typedef Model::state_error_variance_row model_state_error_variance_row

  Type of a row of the background error variance.

- typedef Model::state model_state
Type of the model state vector.
- `typedef Model::matrix_state_observation matrix_state_observation`
  Type of the model/observation crossed matrix.
- `typedef Model::state_error_variance model_state_error_variance`
  Type of the background error variance.
- `typedef Model::tangent_linear_operator model_tangent_linear_operator`
  Type of the tangent linear model.
- `typedef ObservationManager::tangent_linear_operator observation_tangent_linear_operator`
  Type of the tangent linear observation operator.
- `typedef ObservationManager::error_variance observation_error_variance`
  Type of the observation error variance.
- `typedef ObservationManager::tangent_linear_operator_row observation_tangent_linear_operator_row`
  Type of a row of the tangent linear observation operator.
- `typedef ObservationManager::observation observation`
  Type of the observation vector.

Public Member Functions

- `FourDimensionalVariational ()`
  Main constructor.
- `~FourDimensionalVariational ()`
  Destructor.
- `void Initialize (string configuration_file, bool initialize_model=true, bool initialize_observation_manager=true)`
  Initializes the driver.
- `void Initialize (VerdandiOps &configuration, bool initialize_model=true, bool initialize_observation_manager=true)`
  Initializes the driver.
- `void InitializeStep ()`
  Initializes a step for the extended Kalman filter.
- `void Forward ()`
  Performs a step forward, with optimal interpolation at the end.
- `void Analyze ()`
  Computes an analysis.
- `void FinalizeStep ()`
  Finalizes a step for the model.
- `void Finalize ()`
  Finalizes the model.
- `bool HasFinished ()`
  Checks whether the model has finished.
- `Model & GetModel ()`
  Returns the model.
- `ObservationManager & GetObservationManager ()`
  Returns the observation manager.
- `OutputSaver & GetOutputSaver ()`
  Returns the output saver.
- `string GetName () const`
  Returns the name of the class.
- `void Message (string message)`
  Receives and handles a message.
- `Ts Cost (const model_state &x, model_state &gradient)`
  Generates on Mon Sep 28 2015 16:49:58 for Verdandi by Doxygen
Cost function.
- Ts Constraint (const model_state &x, model_state &gradient)
  Constraint function.
- void SetInitialTime (double time)
  Sets initial time.

Static Public Member Functions
- static Ts StaticCost (const model_state &x, model_state &gradient, void *object)
  Static cost function.
- static Ts StaticConstraint (const model_state &x, model_state &gradient, void *object)
  Constraints.
- static void StaticMessage (void *object, string message)
  Receives and handles a message with a static method.

Protected Attributes
- Model model_
  Underlying model.
- double initial_time_
  Initial time.
- ObservationManager observation_manager_
  Observation manager.
- int iteration_
  Iteration.
- string configuration_file_
  Path to the configuration file.
- string model_configuration_file_
  Path to the model configuration file.
- string observation_configuration_file_
  Path to the configuration file for the observation manager.
- map<string, bool> option_display_
  Display options.
- int Nstate_
  Dimension of the state.
- int Nobservation_
  Number of observations.
- bool analyze_first_step_
  Should an analysis be computed at the first step?
- int Ncall_cost_
- string observation_tangent_linear_operator_access_
  Method of access to the tangent linear operator: "element" or "matrix".
- Optimization optimization_
- model_state state_first_guess_
- OutputSaver output_saver_
  Output saver.

48.19.1 Detailed Description

This class implements 4D-Var.
48.19.2 Constructor & Destructor Documentation

48.19.2.1 Verdandi::FourDimensionalVariational::FourDimensionalVariational ( )

Main constructor.
Builds the driver and reads option keys in the configuration file.

Parameters

| in         | configuration_file | configuration file |

Definition at line 48 of file FourDimensionalVariational.cxx.

48.19.3 Member Function Documentation

48.19.3.1 Model & Verdandi::FourDimensionalVariational::GetModel ( )

Returns the model.

Returns

The model.

Definition at line 383 of file FourDimensionalVariational.cxx.

48.19.3.2 string Verdandi::FourDimensionalVariational::GetName ( ) const [virtual]

Returns the name of the class.

Returns

The name of the class.

Reimplemented from Verdandi::VerdandiBase.

Definition at line 425 of file FourDimensionalVariational.cxx.

48.19.3.3 ObservationManager & Verdandi::FourDimensionalVariational::GetObservationManager ( )

Returns the observation manager.

Returns

The observation manager.

Definition at line 397 of file FourDimensionalVariational.cxx.

48.19.3.4 OutputSaver & Verdandi::FourDimensionalVariational::GetOutputSaver ( )

Returns the output saver.

Returns

The output saver.

Definition at line 411 of file FourDimensionalVariational.cxx.
48.19.3.5 bool Verdandi::FourDimensionalVariational::HasFinished ( )

Checks whether the model has finished.

Returns
True if no more data assimilation is required, false otherwise.

Definition at line 369 of file FourDimensionalVariational.cxx.

48.19.3.6 void Verdandi::FourDimensionalVariational::Initialize ( string configuration_file, bool initialize_model = true, bool initialize_observation_manager = true )

Initializes the driver.
Initializes the model and the observation manager. Optionally computes the analysis of the first step.
Definition at line 85 of file FourDimensionalVariational.cxx.

48.19.3.7 void Verdandi::FourDimensionalVariational::Initialize ( VerdandiOps & configuration, bool initialize_model = true, bool initialize_observation_manager = true )

Initializes the driver.
Initializes the model and the observation manager. Optionally computes the analysis of the first step.
Definition at line 101 of file FourDimensionalVariational.cxx.

48.19.3.8 void Verdandi::FourDimensionalVariational::InitializeStep ( )

Initializes a step for the extended Kalman filter.
Initializes a step for the model.
Definition at line 260 of file FourDimensionalVariational.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/FourDimensionalVariational.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/FourDimensionalVariational.cxx

48.20 Verdandi::GridToNetworkObservationManager Class Reference

Observation operator that maps from a grid to a network.

#include <GridToNetworkObservationManager.hxx>

Inheritance diagram for Verdandi::GridToNetworkObservationManager:

```
Verdandi::VerdandiBase

Verdandi::GridToNetworkObservationManager
```

Public Types

- typedef Matrix< T, General, RowSparse > tangent_linear_operator
Type of the tangent linear operator.
- `typedef Matrix< T, General, RowSparse > error_variance`
  Type of the observation error covariance matrix.
- `typedef Vector< T > tangent_linear_operator_row`
  Type of a row of the tangent linear operator.
- `typedef Vector< T > observation`
  Type of the observation vector.
- `typedef Vector< T > observation_vector`
  Type of the observation vector.
- `typedef Vector2< T > observation_vector2`
  Type of the observation vector 2.
- `typedef Vector3< T > observation_vector3`
  Type of the observation vector 3.
- `typedef Vector< int > variable_vector`
  Type of the variable vector.
- `typedef Vector2< int > variable_vector2`
  Type of the variable vector 2.
- `typedef Vector3< int > variable_vector3`
  Type of the variable vector 3.
- `typedef Vector< int > index_vector`
  Type of the index vector.
- `typedef Vector2< int > index_vector2`
  Type of the index vector 2.
- `typedef Vector3< int > index_vector3`
  Type of the index vector 3.
- `typedef Vector< double > time_vector`
  Type of the time vector.
- `typedef Vector2< double > time_vector2`
  Type of the time vector 2.
- `typedef Vector3< double > time_vector3`
  Type of the time vector 3.

Public Member Functions

- `GridToNetworkObservationManager ()`
  Default constructor.
- `template< class Model > GridToNetworkObservationManager (Model &model, string configuration_file)`
  Main constructor.
- `~GridToNetworkObservationManager ()`
  Destructor.
- `template< class Model > void Initialize (Model &model, string configuration_file)`
  Initializes the observation manager.
- `void DiscardObservation (bool discard_observation)`
  Activates or deactivates the option 'discard_observation'.
- `void SetAllActive ()`
  Sets all observation locations as active.
- `int CreateTrack ()`
  Creates a new track.
- `void SetTrack (int track)`
Sets the track to a given track.

- template<class Model>
  
  void SetTime (Model &model, double time)
  
  Sets the time of observations to be loaded.

- void SetTime (double time)
  
  Sets the time of observations to be loaded.

- void SetAvailableTime (double time, time_vector &available_time) const
  
  Sets the available observation times at a given time.

- void SetAvailableTime (double time_inf, double time_sup, time_vector &available_time) const
  
  Sets available observation times at a given time interval.

- void GetFlattenedObservation (double time, observation_vector &observation)
  
  Gets observations flattened over a list of times.

- void GetFlattenedObservation (double time_inf, double time_sup, observation_vector &observation)
  
  Gets observations flattened over a list of times.

- void GetFlattenedObservation (observation_vector &observation)
  
  Gets observations flattened over a list of times.

- void GetFlattenedObservation (const time_vector &available_time, observation_vector &observation)
  
  Gets observations flattened over a list of times.

- void GetFlattenedObservation (double time, variable_vector &observation_variable, observation_vector &observation)
  
  Gets observations flattened over a list of times.

- void GetFlattenedObservation (double time_inf, double time_sup, variable_vector &observation_variable, observation_vector &observation)
  
  Gets observations flattened over a list of times.

- void GetFlattenedObservation (variable_vector &observation_variable, observation_vector &observation)
  
  Gets observations flattened over a list of times.

- void GetFlattenedObservation (const time_vector &available_time, variable_vector &observation_variable, observation_vector &observation)
  
  Gets observations flattened over a list of times.

- void GetAggregatedObservation (double time, observation_vector &observation)
  
  Gets observations aggregated over a list of times.

- void GetAggregatedObservation (double time_inf, double time_sup, observation_vector &observation)
  
  Gets observations aggregated over a list of times.

- void GetAggregatedObservation (observation_vector &observation)
  
  Gets observations aggregated over a list of times.

- void GetAggregatedObservation (const time_vector &available_time, observation_vector &observation)
  
  Gets observations aggregated over a list of times.

- void GetAggregatedObservation (double time, variable_vector &observation_variable, observation_vector &observation)
  
  Gets observations aggregated over a list of times.

- void GetAggregatedObservation (double time_inf, double time_sup, variable_vector &observation_variable, observation_vector &observation)
  
  Gets observations aggregated over a list of times.

- void GetAggregatedObservation (variable_vector &observation_variable, observation_vector &observation)
  
  Gets observations aggregated over a list of times.
- void GetAggregatedObservation (double time_inf, double time_sup, variable_vector &observation_variable, observation_vector2 &observation2)
  Gets observations aggregated over a list of times.
- void GetAggregatedObservation (variable_vector &observation_variable, observation_vector2 &observation2)
  Gets observations aggregated over a list of times.
- void GetAggregatedObservation (const time_vector &available_time, variable_vector &observation_variable, observation_vector2 &observation2)
  Gets observations aggregated over a list of times.
- void GetAggregatedObservation (double time, variable_vector &observation_variable, index_vector2 &observation_index2, observation_vector2 &observation2)
  Gets observations aggregated over a list of times.
- void GetAggregatedObservation (double time_inf, double time_sup, variable_vector &observation_variable, index_vector2 &observation_index2, observation_vector2 &observation2)
  Gets observations aggregated over a list of times.
- void GetAggregatedObservation (variable_vector &observation_variable, index_vector2 &observation_index2, observation_vector2 &observation2)
  Gets observations aggregated over a list of times.
- void GetAggregatedObservation (const time_vector &available_time, variable_vector &observation_variable, index_vector2 &observation_index2, observation_vector2 &observation2)
  Gets observations aggregated over a list of times.
- void GetRawObservation (double time, observation_vector2 &observation2)
  Gets available observations at a given time.
- void GetRawObservation (double time_inf, double time_sup, observation_vector2 &observation2)
  Gets available observations in a given interval.
- void GetRawObservation (observation_vector2 &observation2)
  Gets observations at the current time.
- void GetRawObservation (const time_vector &available_time, observation_vector2 &observation2)
  Gets observations of a list of times.
- void GetRawObservation (double time, variable_vector2 &observation_variable2, observation_vector3 &observation3)
  Gets available observations at a given time.
- void GetRawObservation (double time_inf, double time_sup, variable_vector2 &observation_variable2, observation_vector3 &observation3)
  Gets available observations in a given interval.
- void GetRawObservation (variable_vector2 &observation_variable2, observation_vector3 &observation3)
  Gets observations at the current time.
- void GetRawObservation (const time_vector &available_time, variable_vector2 &observation_variable2, observation_vector3 &observation3)
  Gets observations of a list of times.
- void GetRawObservation (double time, variable_vector2 &observation_variable2, index_vector3 &observation_index3, observation_vector3 &observation3)
  Gets available observations at a given time.
- void GetRawObservation (double time_inf, double time_sup, variable_vector2 &observation_variable2, index_vector3 &observation_index3, observation_vector3 &observation3)
  Gets available observations in a given interval.
- void GetRawObservation (variable_vector2 &observation_variable2, index_vector3 &observation_index3, observation_vector3 &observation3)
  Gets observations at the current time.
- void GetRawObservation (const time_vector &available_time, variable_vector2 &observation_variable2, index_vector3 &observation_index3, observation_vector3 &observation3)
  Gets observations of a list of times.
- **void** ReadObservationVariable (const time_vector &available_time, variable_vector2 &observation_variable2) const
  
  *Builds variables vector associated with given observations.*

- **void** ReadObservation (const time_vector &available_time, const variable_vector2 &observation_variable2, observation_vector3 &observation3) const
  
  *Builds observations associated with given times and variables.*

- **void** ReadObservation (const time_vector &available_time, observation_vector2 &observation2) const
  
  *Builds observations associated with given times.*

- **void** ReadObservation (double time, int variable, observation_vector &observation) const
  
  *Reads observation from observation file given a time and a variable.*

- **void** ReadObservationIndex (const time_vector &available_time, const variable_vector2 &observation_variable2, index_vector3 &observation_index3) const
  
  *Reads observations indexes.*

- **observation &** GetObservation ()
  
  *Gets observation.*

- **template**<class state >
  
  **observation &** GetInnovation (const state &x)
  
  *Gets innovation.*

- **bool** HasObservation () const
  
  *Indicates if some observations are available at current time.*

- **int** GetNobservation () const
  
  *Gets Nobservation_value.*

- **bool** IsOperatorSparse () const
  
  *Checks whether the observation operator is available in a sparse matrix.*

- **bool** IsErrorSparse () const
  
  *Checks whether the observation error covariance matrix is sparse.*

- **bool** HasErrorMatrix () const
  
  *Checks whether the observation error covariance is available in a matrix.*

- **template**<class state >
  
  **void** ApplyOperator (const state &x, observation &y) const
  
  *Applies the operator to a given vector.*

- **template**<class state >
  
  **void** ApplyTangentLinearOperator (const state &x, observation &y) const
  
  *Applies the tangent linear operator to a given vector.*

- **T** GetTangentLinearOperator (int i, int j) const
  
  *Linearized observation operator.*

- **tangent_linear_operator_row &** GetTangentLinearOperatorRow (int row)
  
  *Linearized observation operator.*

- **const tangent_linear_operator &** GetTangentLinearOperator () const
  
  *Linearized observation operator.*

- **template**<class state >
  
  **void** ApplyAdjointOperator (const state &x, observation &y) const
  
  *Applies the adjoint operator to a given vector.*

- **bool** HasBLUECorrection () const
  
  *Checks whether a BLUE correction is available.*

- **void** GetBLUECorrection (Vector< T > &BLUE_correction) const
  
  *Gets the BLUE correction.*

- **T** GetErrorVariance (int i, int j) const
  
  *Observation error covariance.*

- **const error_variance &** GetErrorVariance () const
  
  *Observation error covariance matrix.*

- **const error_variance &** GetErrorVarianceInverse () const
Returns the inverse of the observation error covariance matrix.

- string GetName () const
  Returns the name of the class.
- void Message (string message)
  Receives and handles a message.

Static Public Member Functions

- static void StaticMessage (void ∗object, string message)
  Receives and handles a message with a static method.

Protected Attributes

- string observation_file_
  File that stores the observations.
- string observation_type_
  How are stored the observations.
- int Nobservation_
  Number total of observations at current time.
- size_t Nbyte_observation_
  Size in byte of an observations vector.
- double Delta_t_
  Period with which observations are available.
- int Nskip_
  Period with which available observations are actually loaded.
- double final_time_
  Duration during which observations are assimilated.
- string observation_storage_
  How are read the observations.
- double time_
  Requested time.
- time_vector available_time_
  Available observation time of the time interval.
- Vector<double > contribution_
  Contribution associated with available observations.
- OBSERVATION_AGGREGATOR<T > observation_aggregator_
  Observations aggregator.
- Vector<T > location_x_
  Index along x.
- Vector<T > location_y_
  Index along y.
- Matrix<int > interpolation_index_
  Interpolation indices for all locations.
- Matrix<T > interpolation_weight_
  Interpolation weights for all locations.
- Matrix<int > active_interpolation_index_
  Interpolation indices for active locations.
- Matrix<T > active_interpolation_weight_
  Interpolation weights for active locations.
- T error_variance_value_
Class Documentation

Observation error variance.

- int Nstate_model_
The size of a model state.
- int Nx_model_
  Number of points along x in the model.
- int Ny_model_
  Number of points along y in the model.
- observation observation_
  Observation currently stored.
- observation innovation_
  Innovation currently stored.
- int current_row_
  Index of the row of H currently stored.
- tangent_linear_operator_row tangent_operator_row_
  Value of the row of H currently stored.

48.20.1 Detailed Description

Observation operator that maps from a grid to a network.
The operator linearly interpolates from a regular grid to a list of locations (i.e., a network).

Template Parameters

| T the type of floating-point numbers. |

48.20.2 Constructor & Destructor Documentation

48.20.2.1 Verdandi::GridToNetworkObservationManager::GridToNetworkObservationManager ( )

Default constructor.
It entirely defines the operator: no dimension or size is associated with this implementation.
Definition at line 45 of file GridToNetworkObservationManager.cxx.

48.20.2.2 template< class Model > Verdandi::GridToNetworkObservationManager::GridToNetworkObservationManager ( Model & model, string configuration_file )

Main constructor.
It defines the operator for a 2D regular grid.

Parameters

| in model model. |
| in configuration_file configuration_file. |

Template Parameters

| Model the model type; e.g. ShallowWater<double> |

Definition at line 59 of file GridToNetworkObservationManager.cxx.
48.20.3 Member Function Documentation

48.20.3.1 template<class state> void Verdandi::GridToNetworkObservationManager::ApplyAdjointOperator ( const state & x, observation & y ) const
Applies the adjoint operator to a given vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x</th>
<th>a vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>y</td>
<td>the value of the operator at x. It is resized if needed.</td>
</tr>
</tbody>
</table>

Definition at line 1542 of file GridToNetworkObservationManager.cxx.

48.20.3.2 template<class state> void Verdandi::GridToNetworkObservationManager::ApplyOperator ( const state & x, observation & y ) const
Applies the operator to a given vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x</th>
<th>a vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>y</td>
<td>the value of the operator at x. It is resized if needed.</td>
</tr>
</tbody>
</table>

Definition at line 1440 of file GridToNetworkObservationManager.cxx.

48.20.3.3 template<class state> void Verdandi::GridToNetworkObservationManager::ApplyTangentLinearOperator ( const state & x, observation & y ) const
Applies the tangent linear operator to a given vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x</th>
<th>a vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>y</td>
<td>the value of the tangent linear operator at x. It is resized if needed.</td>
</tr>
</tbody>
</table>

Definition at line 1465 of file GridToNetworkObservationManager.cxx.

48.20.3.4 int Verdandi::GridToNetworkObservationManager::CreateTrack ( )
Creates a new track.

Returns
The index of the new track.

Definition at line 248 of file GridToNetworkObservationManager.cxx.

48.20.3.5 void Verdandi::GridToNetworkObservationManager::DiscardObservation ( bool discard_observation )
Activates or deactivates the option ‘discard_observation’.

Parameters

| in     | discard_observation | if set to true, each observation will be used at most one time. |
48.20.3.6 void Verdandi::GridToNetworkObservationManager::GetAggregatedObservation ( double time, observation_vector & observation )

Gets observations aggregated over a list of times.
The observations available at the given time are loaded and aggregated.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>the given time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation</td>
<td>the aggregated observations.</td>
</tr>
</tbody>
</table>

Definition at line 616 of file GridToNetworkObservationManager.cxx.

48.20.3.7 void Verdandi::GridToNetworkObservationManager::GetAggregatedObservation ( double time_inf, double time_sup, observation_vector & observation )

Gets observations aggregated over a list of times.
The observations in the interval \([time_{inf}, time_{sup}]\) are loaded and aggregated.

Parameters

| in   | time_{inf} | lower bound of the given interval. |
|------| time_{sup} | upper bound (excluded) of the given interval. |
| out  | observation | the aggregated observations. |

Definition at line 634 of file GridToNetworkObservationManager.cxx.

48.20.3.8 void Verdandi::GridToNetworkObservationManager::GetAggregatedObservation ( observation_vector & observation )

Gets observations aggregated over a list of times.
The observations available are loaded and aggregated.

Parameters

| out | observation | the aggregated observations. |

Definition at line 650 of file GridToNetworkObservationManager.cxx.

48.20.3.9 void Verdandi::GridToNetworkObservationManager::GetAggregatedObservation ( const time_vector & available_time, observation_vector & observation )

Gets observations aggregated over a list of times.
The observations available at the times \(available\_time\) are loaded and aggregated.

Parameters

| in   | available_time | the given observation time vector. |
|------| observation    | the aggregated observations. |

Definition at line 664 of file GridToNetworkObservationManager.cxx.
48.20.3.10 void Verdandi::GridToNetworkObservationManager::GetAggregatedObservation ( double time, variable_vector & observation_variable, observation_vector2 & observation2 )

Gets observations aggregated over a list of times.
The observations available at the given time are loaded and aggregated.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td></td>
<td>time</td>
</tr>
<tr>
<td>out</td>
<td>observation_variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation2</td>
<td>the aggregated observations.</td>
</tr>
</tbody>
</table>

Definition at line 687 of file GridToNetworkObservationManager.cxx.

48.20.3.11 void Verdandi::GridToNetworkObservationManager::GetAggregatedObservation ( double time_inf, double time_sup, variable_vector & observation_variable, observation_vector2 & observation2 )

Gets observations aggregated over a list of times.
The observations in the interval \([\text{time}_\text{inf}, \text{time}_\text{sup}]\) are loaded and aggregated.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>time_inf</td>
<td>lower bound of the given interval.</td>
</tr>
<tr>
<td>in</td>
<td>time_sup</td>
<td>upper bound (excluded) of the given interval.</td>
</tr>
<tr>
<td>out</td>
<td>observation_variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation2</td>
<td>the aggregated observations.</td>
</tr>
</tbody>
</table>

Definition at line 709 of file GridToNetworkObservationManager.cxx.

48.20.3.12 void Verdandi::GridToNetworkObservationManager::GetAggregatedObservation ( variable_vector & observation_variable, observation_vector2 & observation2 )

Gets observations aggregated over a list of times.
The observations available are loaded and aggregated.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation_variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation2</td>
<td>the aggregated observations.</td>
</tr>
</tbody>
</table>

Definition at line 729 of file GridToNetworkObservationManager.cxx.

48.20.3.13 void Verdandi::GridToNetworkObservationManager::GetAggregatedObservation ( const time_vector & available_time, variable_vector & observation_variable, observation_vector2 & observation2 )

Gets observations aggregated over a list of times.
The observations available at the times \(\text{available}_\text{time}\) are loaded and aggregated.
### Class Documentation

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>available_time</th>
<th>the given observation time vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation_variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation2</td>
<td>the aggregated observations.</td>
</tr>
</tbody>
</table>

Definition at line 747 of file GridToNetworkObservationManager.cxx.

#### 48.20.3.14

```cpp
void Verdandi::GridToNetworkObservationManager::GetAggregatedObservation ( double time, variable_vector & observation_variable, index_vector2 & observation_index2, observation_vector2 & observation2 )
```

Gets observations aggregated over a list of times.

The observations available at the given time are loaded and aggregated.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>the given time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation_variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation_index2</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation2</td>
<td>the aggregated observations.</td>
</tr>
</tbody>
</table>

Definition at line 779 of file GridToNetworkObservationManager.cxx.

#### 48.20.3.15

```cpp
void Verdandi::GridToNetworkObservationManager::GetAggregatedObservation ( double time_inf, double time_sup, variable_vector & observation_variable, index_vector2 & observation_index2, observation_vector2 & observation2 )
```

Gets observations aggregated over a list of times.

The observations in the interval \([time_inf, time_sup]\) are loaded and aggregated.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>time_inf</th>
<th>lower bound of the given interval.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>time_sup</td>
<td>upper bound (excluded) of the given interval.</td>
</tr>
<tr>
<td>out</td>
<td>observation_variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation_index2</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation2</td>
<td>the aggregated observations.</td>
</tr>
</tbody>
</table>

Definition at line 803 of file GridToNetworkObservationManager.cxx.

#### 48.20.3.16

```cpp
void Verdandi::GridToNetworkObservationManager::GetAggregatedObservation ( variable_vector & observation_variable, index_vector2 & observation_index2, observation_vector2 & observation2 )
```

Gets observations aggregated over a list of times.

The observations available are loaded and aggregated.

**Parameters**

| out         | observation_variable | variables associated with the observations. |
## Verdandi::GridToNetworkObservationManager Class Reference

### 48.20.3.17 void Verdandi::GridToNetworkObservationManager::GetAggregatedObservation ( const time_vector & available_time, variable_vector & observation_variable, index_vector2 & observation_index2, observation_vector2 & observation2 )

Gets observations aggregated over a list of times. The observations available at the times *available_time* are loaded and aggregated.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>available_time</th>
<th>the given observation time vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation_variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation_index2</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation2</td>
<td>the aggregated observations.</td>
</tr>
</tbody>
</table>

Definition at line 845 of file GridToNetworkObservationManager.cxx.

### 48.20.3.18 void Verdandi::GridToNetworkObservationManager::GetBLUECorrection ( Vector<T> & BLUE_correction ) const

Gets the BLUE correction.

**Parameters**

| out | BLUE_correction | BLUE correction vector. |

Definition at line 1578 of file GridToNetworkObservationManager.cxx.

### 48.20.3.19 T Verdandi::GridToNetworkObservationManager::GetErrorVariance ( int i, int j ) const

Observation error covariance.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>i</th>
<th>row index.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>j</td>
<td>column index.</td>
</tr>
</tbody>
</table>

**Returns**

The element \((i, j)\) of the observation error covariance.

Definition at line 1593 of file GridToNetworkObservationManager.cxx.

### 48.20.3.20 const GridToNetworkObservationManager<T>::error_variance & Verdandi::GridToNetworkObservationManager::GetErrorVariance ( ) const

Observation error covariance matrix.
Returns
The matrix of the observation error covariance.

Definition at line 1610 of file GridToNetworkObservationManager.cxx.

48.20.3.21 const GridToNetworkObservationManager\< T \>::error_variance & Verdandi::GridToNetworkObservationManager::GetErrorVarianceInverse ( ) const

Returns the inverse of the observation error covariance matrix.

Returns
The inverse of the matrix of the observation error covariance.

Definition at line 1624 of file GridToNetworkObservationManager.cxx.

48.20.3.22 void Verdandi::GridToNetworkObservationManager::GetFlattenedObservation ( double time, observation_vector & observation )

Gets observations flattened over a list of times.
The observations available at time time are loaded and concatenated in a vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>the given time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation</td>
<td>the observation to be flattened.</td>
</tr>
</tbody>
</table>

Definition at line 364 of file GridToNetworkObservationManager.cxx.

48.20.3.23 void Verdandi::GridToNetworkObservationManager::GetFlattenedObservation ( double time_inf, double time_sup, observation_vector & observation )

Gets observations flattened over a list of times.
The observations in the interval [time_inf, time_sup] are loaded and concatenated in a vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time_inf</th>
<th>lower bound of the given interval.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>time_sup</td>
<td>upper bound (excluded) of the given interval.</td>
</tr>
<tr>
<td>out</td>
<td>observation</td>
<td>the observation to be flattened.</td>
</tr>
</tbody>
</table>

Definition at line 382 of file GridToNetworkObservationManager.cxx.

48.20.3.24 void Verdandi::GridToNetworkObservationManager::GetFlattenedObservation ( observation_vector & observation )

Gets observations flattened over a list of times.
The observations available are loaded and concatenated in a vector.

Parameters

| out        | observation | the observation to be loaded. |

Definition at line 397 of file GridToNetworkObservationManager.cxx.
void Verdandi::GridToNetworkObservationManager::GetFlattenedObservation ( const time_vector & available_time, observation_vector & observation )

Gets observations flattened over a list of times.

The observations available at the times available_time are loaded and concatenated in a vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>available_time</th>
<th>the given observation time vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 411 of file GridToNetworkObservationManager.cxx.

void Verdandi::GridToNetworkObservationManager::GetFlattenedObservation ( double time, variable_vector & observation_variable, observation_vector & observation )

Gets observations flattened over a list of times.

The observations available at time time are loaded and concatenated in a vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>the given time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation_variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation</td>
<td>the observation to be flattened.</td>
</tr>
</tbody>
</table>

Definition at line 433 of file GridToNetworkObservationManager.cxx.

void Verdandi::GridToNetworkObservationManager::GetFlattenedObservation ( double time_inf, double time_sup, variable_vector & observation_variable, observation_vector & observation )

 Gets observations flattened over a list of times.

The observations in the interval [time_inf, time_sup] are loaded and concatenated in a vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time_inf</th>
<th>lower bound of the given interval.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>time_sup</td>
<td>upper bound (excluded) of the given interval.</td>
</tr>
<tr>
<td>out</td>
<td>observation_variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation</td>
<td>the observation to be flattened.</td>
</tr>
</tbody>
</table>

Definition at line 455 of file GridToNetworkObservationManager.cxx.

void Verdandi::GridToNetworkObservationManager::GetFlattenedObservation ( variable_vector & observation_variable, observation_vector & observation )

Gets observations flattened over a list of times.

The observations available are loaded and concatenated in a vector.

Parameters

| out | observation_variable | variables associated with the observations. |
| out | observation | the observation to be loaded. |
48.20.3.29 void Verdandi::GridToNetworkObservationManager::GetFlattenedObservation ( const time_vector & available_time, variable_vector & observation_variable, observation_vector & observation )

Gets observations flattened over a list of times.
The observations available at the times available_time are loaded and concatenated in a vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>available_time</th>
<th>the given observation time vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation_variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

48.20.3.30 void Verdandi::GridToNetworkObservationManager::GetFlattenedObservation ( double time, variable_vector & observation_variable, index_vector & observation_index, observation_vector & observation )

Gets observations flattened over a list of times.
The observations available at time time are loaded and concatenated in a vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>the given time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation_variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation_index</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation</td>
<td>the observation to be flattened.</td>
</tr>
</tbody>
</table>

48.20.3.31 void Verdandi::GridToNetworkObservationManager::GetFlattenedObservation ( double time_inf, double time_sup, variable_vector & observation_variable, index_vector & observation_index, observation_vector & observation )

Gets observations flattened over a list of times.
The observations in the interval [time_inf, time_sup] are loaded and concatenated in a vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time_inf</th>
<th>lower bound of the given interval.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>time_sup</td>
<td>upper bound (excluded) of the given interval.</td>
</tr>
<tr>
<td>out</td>
<td>observation_variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation_index</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation</td>
<td>the observation to be flattened.</td>
</tr>
</tbody>
</table>
Gets observations flattened over a list of times.
The observations available are loaded and concatenated in a vector.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>observation_variable</th>
<th>variables associated with the observations.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation_index</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 564 of file GridToNetworkObservationManager.cxx.

Gets observations flattened over a list of times.
The observations available at the times available_time are loaded and concatenated in a vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>available_time</th>
<th>the given observation time vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation_variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation_index</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 584 of file GridToNetworkObservationManager.cxx.

Gets innovation.

Parameters

| in    | state | state vector. |

Returns

The innovation vector.

Definition at line 1340 of file GridToNetworkObservationManager.cxx.

Returns the name of the class.
Returns

The name of the class.

Reimplemented from Verdandi::VerdandiBase.

Definition at line 1639 of file GridToNetworkObservationManager.cxx.

48.20.3.36 int Verdandi::GridToNetworkObservationManager::GetNobservation ( ) const

Gets Nobservation_value.

Returns

The total number of observation at current time.

Definition at line 1369 of file GridToNetworkObservationManager.cxx.

48.20.3.37 GridToNetworkObservationManager\langle T \rangle::observation & Verdandi::GridToNetworkObservationManager\::GetObservation ( )

Gets observation.

Returns

The observation vector.

Definition at line 1320 of file GridToNetworkObservationManager.cxx.

48.20.3.38 void Verdandi::GridToNetworkObservationManager::GetRawObservation ( double time, observation_vector2 & observation2 )

Gets available observations at a given time.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>the given time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation2</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 882 of file GridToNetworkObservationManager.cxx.

48.20.3.39 void Verdandi::GridToNetworkObservationManager::GetRawObservation ( double time_inf, double time_sup, observation_vector2 & observation2 )

Gets observations available in a given interval.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time_inf</th>
<th>lower bound of the given interval.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>time_sup</td>
<td>upper bound (excluded) of the given interval.</td>
</tr>
<tr>
<td>out</td>
<td>observation2</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 899 of file GridToNetworkObservationManager.cxx.
void Verdandi::GridToNetworkObservationManager::GetRawObservation ( observation_vector2 & observation2 )

Gets observations at the current time.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>observation2</th>
<th>the observation to be loaded.</th>
</tr>
</thead>
</table>

Definition at line 914 of file GridToNetworkObservationManager.cxx.

void Verdandi::GridToNetworkObservationManager::GetRawObservation ( const time_vector & available_time, observation_vector2 & observation2 )

Gets observations of a list of times.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>available_time</th>
<th>the given observation time vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation2</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 927 of file GridToNetworkObservationManager.cxx.

void Verdandi::GridToNetworkObservationManager::GetRawObservation ( double time, variable_vector2 & observation_variable2, observation_vector3 & observation3 )

Gets available observations at a given time.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>the given time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation_variable2</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation3</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 946 of file GridToNetworkObservationManager.cxx.

void Verdandi::GridToNetworkObservationManager::GetRawObservation ( double time_inf, double time_sup, variable_vector2 & observation_variable2, observation_vector3 & observation3 )

Gets observations available in a given interval.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time_inf</th>
<th>lower bound of the given interval.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>time_sup</td>
<td>upper bound (excluded) of the given interval.</td>
</tr>
<tr>
<td>out</td>
<td>observation_variable2</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation3</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 967 of file GridToNetworkObservationManager.cxx.

void Verdandi::GridToNetworkObservationManager::GetRawObservation ( variable_vector2 & observation_variable2, observation_vector3 & observation3 )

Gets observations at the current time.
### Parameters

<table>
<thead>
<tr>
<th>Direction</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation variable 2</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation 3</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 987 of file `GridToNetworkObservationManager.cxx`.

#### 48.20.3.45 void Verdandi::GridToNetworkObservationManager::GetRawObservation ( const time_vector & available_time, variable_vector2 & observation_variable2, observation_vector3 & observation3 )

Gets observations of a list of times.

### Parameters

<table>
<thead>
<tr>
<th>Direction</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>available_time</td>
<td>the given observation time vector.</td>
</tr>
<tr>
<td>out</td>
<td>observation variable 2</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation 3</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 1004 of file `GridToNetworkObservationManager.cxx`.

#### 48.20.3.46 void Verdandi::GridToNetworkObservationManager::GetRawObservation ( double time, variable_vector2 & observation_variable2, index_vector3 & observation_index3, observation_vector3 & observation3 )

Gets available observations at a given time.

### Parameters

<table>
<thead>
<tr>
<th>Direction</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>time</td>
<td>the given time.</td>
</tr>
<tr>
<td>out</td>
<td>observation variable 2</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation index 3</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation 3</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 1026 of file `GridToNetworkObservationManager.cxx`.

#### 48.20.3.47 void Verdandi::GridToNetworkObservationManager::GetRawObservation ( double time_inf, double time_sup, variable_vector2 & observation_variable2, index_vector3 & observation_index3, observation_vector3 & observation3 )

Gets observations available in a given interval.

### Parameters

<table>
<thead>
<tr>
<th>Direction</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>time_inf</td>
<td>lower bound of the given interval.</td>
</tr>
<tr>
<td>in</td>
<td>time_sup</td>
<td>upper bound (excluded) of the given interval.</td>
</tr>
<tr>
<td>out</td>
<td>observation variable 2</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation index 3</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation 3</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 1049 of file `GridToNetworkObservationManager.cxx`. 

---

Generated on Mon Sep 28 2015 16:49:58 for Verdandi by Doxygen
void Verdandi::GridToNetworkObservationManager::GetRawObservation ( variable_vector2 & observation2, index_vector3 & observation_index3, observation_vector3 & observation3 )

Gets observations at the current time.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>observation_ variable2</th>
<th>variables associated with the observations.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation_ index3</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation3</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 1071 of file GridToNetworkObservationManager.cxx.

void Verdandi::GridToNetworkObservationManager::GetRawObservation ( const time_vector & available_time, variable_vector2 & observation_variable2, index_vector3 & observation_index3, observation_vector3 & observation3 )

Gets observations of a list of times.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>available_time</th>
<th>the given observation time vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation_ variable2</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation_ index3</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation3</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 1090 of file GridToNetworkObservationManager.cxx.

T Verdandi::GridToNetworkObservationManager::GetTangentLinearOperator ( int i, int j ) const

Linearized observation operator.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>i</th>
<th>row index.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>j</td>
<td>column index.</td>
</tr>
</tbody>
</table>

Returns

The element \((i, j)\) of the linearized operator.

Definition at line 1479 of file GridToNetworkObservationManager.cxx.

const GridToNetworkObservationManager< T >::tangent_linear_operator&
Verdandi::GridToNetworkObservationManager::GetTangentLinearOperator ( ) const

Linearized observation operator.

Returns

The matrix of the linearized operator.

Definition at line 1526 of file GridToNetworkObservationManager.cxx.
GridToNetworkObservationManager::GetTangentLinearOperatorRow (int row)

Linearized observation operator.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>row</th>
<th>row index</th>
</tr>
</thead>
</table>

**Returns**

The row row of the linearized operator.

Definition at line 1501 of file GridToNetworkObservationManager.cxx.

Verdandi::GridToNetworkObservationManager::HasBLUECorrection () const

Checks whether a BLUE correction is available.

**Returns**

True if a BLUE correction is available, false otherwise.

Definition at line 1565 of file GridToNetworkObservationManager.cxx.

Verdandi::GridToNetworkObservationManager::HasErrorMatrix () const

Checks whether the observation error covariance is available in a matrix.

**Returns**

True if the observation error covariance is available in a matrix, false otherwise.

Definition at line 1416 of file GridToNetworkObservationManager.cxx.

Verdandi::GridToNetworkObservationManager::Initialize (Model & model, string configuration_file)

Initializes the observation manager.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>model</th>
<th>model.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>configuration_file</td>
<td>configuration file.</td>
</tr>
</tbody>
</table>

**Template Parameters**

| Model | the model type; e.g. ShallowWater<double> |

First abscissa.

First ordinate.

Space step along x.

Space step along y.

Definition at line 88 of file GridToNetworkObservationManager.cxx.
### Verdandi::GridToNetworkObservationManager Class Reference

#### 48.20.3.56 bool Verdandi::GridToNetworkObservationManager::IsErrorSparse ( ) const

Checks whether the observation error covariance matrix is sparse.

**Returns**

True if the observation error covariance matrix is sparse, false otherwise.

Definition at line 1399 of file GridToNetworkObservationManager.cxx.

#### 48.20.3.57 bool Verdandi::GridToNetworkObservationManager::IsOperatorSparse ( ) const

Checks whether the observation operator is available in a sparse matrix.

**Returns**

True if the observation operator is available in a sparse matrix, false otherwise.

Definition at line 1383 of file GridToNetworkObservationManager.cxx.

#### 48.20.3.58 void Verdandi::GridToNetworkObservationManager::ReadObservation ( const time_vector & available_time, const variable_vector2 & observation_variable2, observation_vector3 & observation3 ) const

Builds observations associated with given times and variables.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>available_time</th>
<th>the given observation time vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>observation_ -variable2</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation3</td>
<td>the observations.</td>
</tr>
</tbody>
</table>

Definition at line 1135 of file GridToNetworkObservationManager.cxx.

#### 48.20.3.59 void Verdandi::GridToNetworkObservationManager::ReadObservation ( const time_vector & available_time, observation_vector2 & observation2 ) const

Builds observations associated with given times.

**Parameters**

| in   | available_time | the given observation time vector. |
| out  | observation2 | the observations. |

Definition at line 1162 of file GridToNetworkObservationManager.cxx.

#### 48.20.3.60 void Verdandi::GridToNetworkObservationManager::ReadObservation ( double time, int variable, observation_vector & observation ) const

Reads observation from observation file given a time and a variable.

**Parameters**

| in   | time | the time. |
| in   | variable | the variable. |
| out  | observation | the observations. |
48.20.3.61 void Verdandi::GridToNetworkObservationManager::ReadObservationIndex ( const time_vector & available_time, const variable_vector2 & observation_variable2, index_vector3 & observation_index3 ) const

Reads observations indexes.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>available_time</th>
<th>the available time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>observation_ - variable2</td>
<td>variable associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation_ - index3</td>
<td>the indexes associated with the observations.</td>
</tr>
</tbody>
</table>

48.20.3.62 void Verdandi::GridToNetworkObservationManager::ReadObservationVariable ( const time_vector & available_time, variable_vector2 & observation_variable2 ) const

Builds variables vector associated with given observations.

Parameters

| in | available_time | the given observation time vector. |
| out | observation_ - variable2 | variables associated with the observations. |

48.20.3.63 void Verdandi::GridToNetworkObservationManager::SetAvailableTime ( double time, time_vector & available_time ) const

Sets the available observation times at a given time.

Parameters

| in | time | the given time. |
| out | available_time | the available observation times. |

48.20.3.64 void Verdandi::GridToNetworkObservationManager::SetAvailableTime ( double time_inf, double time_sup, time_vector & available_time ) const

Sets available observation times at a given time interval.

Parameters

| in | time_inf | lower bound of the given time interval. |
| in | time_sup | upper bound (excluded) of the given time interval. |
| out | available_time | the available observation times. |
48.20.3.65 \texttt{template<class Model> \ void Verdandi::GridToNetworkObservationManager::SetTime ( Model & model, double time )}

Sets the time of observations to be loaded.

Parameters

\begin{tabular}{|l|l|}
\hline
in & model & the model. \\
\hline
in & time & a given time. \\
\hline
\end{tabular}

Definition at line 274 of file GridToNetworkObservationManager.cxx.

48.20.3.66 \texttt{void Verdandi::GridToNetworkObservationManager::SetTime ( double time )}

Sets the time of observations to be loaded.

Parameters

\begin{tabular}{|l|l|}
\hline
in & time & a given time. \\
\hline
\end{tabular}

Definition at line 286 of file GridToNetworkObservationManager.cxx.

48.20.3.67 \texttt{void Verdandi::GridToNetworkObservationManager::SetTrack ( int track )}

Sets the track to a given track.

Parameters

\begin{tabular}{|l|l|}
\hline
in & track & the given track. \\
\hline
\end{tabular}

Definition at line 260 of file GridToNetworkObservationManager.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/observation_manager/GridToNetworkObservationManager.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/observation_manager/GridToNetworkObservationManager.cxx

48.21 Verdandi::HamiltonJacobiBellman Class Reference

This class is a solver for Hamilton-Jacobi-Bellman equation.

\texttt{#include <HamiltonJacobiBellman.hxx>}

Inheritance diagram for Verdandi::HamiltonJacobiBellman:

\begin{center}
\begin{tikzpicture}
  \node (V1) {Verdandi::VerdandiBase};
  \node (V2) [below of=V1] {Verdandi::HamiltonJacobiBellman};
  \draw [->] (V1) -- (V2);
\end{tikzpicture}
\end{center}

Public Types

- \texttt{typedef Model::state model\_state}
Public Member Functions

- **HamiltonJacobiBellman ()**
  Main constructor.
- **~HamiltonJacobiBellman ()**
  Destructor.
- **void Initialize (string configuration_file)**
  Initiates the solver.
- **void Initialize (VerdandiOps &configuration)**
  Initiates the solver.
- **void InitializeStep ()**
  Initiates a step for the time integration of HJB equation.
- **void Forward ()**
  Performs a step forward.
- **void AdvectionLxFForward ()**
  Performs a step forward, using a first-order Lax-Friedrichs scheme.
- **void AdvectionBrysonLevyForward ()**
- **void AdvectionGodunov ()**
  Performs a step forward, using a first-order Godunov scheme.
- **void FinalizeStep ()**
  Finalizes a step for the model.
- **void Finalize ()**
  Finalizes the model.
- **T GodunovFlux (T q, T M, T v_l, T v, T v_r) const**
  Computes the Godunov flux along a given dimension.
- **bool HasFinished ()**
  Checks whether the model has finished.
- **Model & GetModel ()**
  Returns the model.
- **ObservationManager & GetObservationManager ()**
  Returns the observation manager.
- **OutputSaver & GetOutputSaver ()**
  Returns the output saver.
- **string GetName () const**
  Returns the name of the class.
- **void Message (string message)**
  Receives and handles a message.

Static Public Member Functions

- **static void StaticMessage (void *object, string message)**
  Receives and handles a message with a static method.
Protected Attributes

- Model model_
  Underlying model.
- ObservationManager observation_manager_
  Observation manager.
- map<string, bool> option_display_
  Display options.
- string configuration_file_
  Path to the configuration file.
- string model_configuration_file_
  Path to the model configuration file.
- string observation_configuration_file_
  Path to the configuration file for the observation manager.
- int Nstate_
  Dimension of the state.
- int Nobservation_
  Number of observations.
- OutputSaver output_saver_
  Output saver.
- bool with_quadratic_term_
  Should the quadratic term be taken into account?
- bool with_advection_term_
  Should the advection term be taken into account?
- bool with_source_term_
  Should the source term be taken into account?
- Matrix<T> Q_0_
  Q_0.
- Vector<T> x_0_
  Position of the minimum of the initial parabola.
- Matrix<T> Q_inv_
  Inverse of Q.
- Matrix<T> R_
  R.
- int Ndimension_
  Number of dimensions.
- int Npoint_
  Total number of points in the domain.
- Vector<T> x_min_
  First coordinate along each dimension.
- Vector<T> Delta_x_
  Space step in each dimension.
- Vector<int> Nx_
  Number of points in each dimension.
- int Nt_
  Number of time steps.
- T initial_time_
  Initial time.
- T Delta_t_
  Time step.
- int iteration_
Class Documentation

Current iteration.
- Vector< T > V_
  Value function $V(t, x)$.
- string scheme_
- Vector< T > a_Delta_x_
  Location of the evolution points in Bryson-Levy scheme.
- bool model_time_dependent_
- Matrix< T > Mx_
  $M(x)$ in every point in space.
- T courant_number_
  Courant number.
- string boundary_condition_type_
- int boundary_condition_index_
- T boundary_condition_
- Vector< T > upper_bound_model_

48.21.1 Detailed Description

This class is a solver for Hamilton-Jacobi-Bellman equation.

48.21.2 Constructor & Destructor Documentation

48.21.2.1 Verdandi::HamiltonJacobiBellman::HamiltonJacobiBellman ( )

Main constructor.
Builds the driver and reads option keys in the configuration file.

Parameters

| in | configuration_file | configuration file |

Definition at line 44 of file HamiltonJacobiBellman.cxx.

48.21.3 Member Function Documentation

48.21.3.1 Model & Verdandi::HamiltonJacobiBellman::GetModel ( )

Returns the model.

Returns

The model.

Definition at line 986 of file HamiltonJacobiBellman.cxx.

48.21.3.2 string Verdandi::HamiltonJacobiBellman::GetName ( ) const [virtual]

Returns the name of the class.
Returns
The name of the class.

Reimplemented from Verdandi::VerdandiBase.
Definition at line 1023 of file HamiltonJacobiBellman.cxx.

48.21.3.3  ObservationManager & Verdandi::HamiltonJacobiBellman::GetObservationManager ( )
Returns the observation manager.

Returns
The observation manager.
Definition at line 999 of file HamiltonJacobiBellman.cxx.

48.21.3.4  OutputSaver & Verdandi::HamiltonJacobiBellman::GetOutputSaver ( )
Returns the output saver.

Returns
The output saver.
Definition at line 1012 of file HamiltonJacobiBellman.cxx.

48.21.3.5  T Verdandi::HamiltonJacobiBellman::GodunovFlux ( T q, T M, T v_l, T v, T v_r ) const [inline]
Computes the Godunov flux along a given dimension.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q )</td>
<td>value of ( Q ) along the given dimension.</td>
</tr>
<tr>
<td>( M )</td>
<td>value of the model ( M(x) ) along the given dimension.</td>
</tr>
<tr>
<td>( v_l )</td>
<td>value in the left cell.</td>
</tr>
<tr>
<td>( v )</td>
<td>value in the center cell.</td>
</tr>
<tr>
<td>( v_r )</td>
<td>value in the right cell.</td>
</tr>
</tbody>
</table>

Returns
The Godunov flux.
Definition at line 944 of file HamiltonJacobiBellman.cxx.

48.21.3.6  bool Verdandi::HamiltonJacobiBellman::HasFinished ( )
Checks whether the model has finished.

Returns
True if no more data assimilation is required, false otherwise.
Definition at line 974 of file HamiltonJacobiBellman.cxx.
48.21.3.7 void Verdandi::HamiltonJacobiBellman::Initialize ( string configuration_file )

Initializes the solver.
Initializes the model and the observation manager. Optionally computes the analysis of the first step.

Parameters

| in | configuration_file | configuration file. |

Definition at line 80 of file HamiltonJacobiBellman.cxx.

48.21.3.8 void Verdandi::HamiltonJacobiBellman::Initialize ( VerdandiOps & configuration )

Initializes the solver.
Initializes the model and the observation manager. Optionally computes the analysis of the first step.

Parameters

| in | configuration_file | configuration file. |

Definition at line 94 of file HamiltonJacobiBellman.cxx.

48.21.3.9 void Verdandi::HamiltonJacobiBellman::InitializeStep ( )

Initializes a step for the time integration of HJB equation.
Initializes a step for the model.
Definition at line 425 of file HamiltonJacobiBellman.cxx.

48.21.4 Member Data Documentation

48.21.4.1 T Verdandi::HamiltonJacobiBellman::boundary_condition_ [protected]

Boundary condition (constant value imposed outside the domain) in case of Dirichlet boundary conditions.
Definition at line 145 of file HamiltonJacobiBellman.hxx.

48.21.4.2 int Verdandi::HamiltonJacobiBellman::boundary_condition_index_ [protected]

0 for Dirichlet boundary conditions, 1 for the extrapolation, and 2 for periodic.
Definition at line 142 of file HamiltonJacobiBellman.hxx.

48.21.4.3 string Verdandi::HamiltonJacobiBellman::boundary_condition_type_ [protected]

Type of the boundary condition (‘Dirichlet’, ‘Extrapolation’ and ‘Period’ are supported).
Definition at line 139 of file HamiltonJacobiBellman.hxx.

48.21.4.4 string Verdandi::HamiltonJacobiBellman::scheme_ [protected]

Name of the numerical scheme: LxF (for first-order Lax-Friedrichs), BrysonLevy (first-order central scheme) or Godunov (first-order).
Definition at line 125 of file HamiltonJacobiBellman.hxx.
48.21.4.5 Vector<T> Verdandi::HamiltonJacobiBellman::upper_bound_model_ [protected]

Upper bound on the absolute value of the model operator, in every direction. Useful in Lax-Friedrichs scheme.
Definition at line 149 of file HamiltonJacobiBellman.hxx.
The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/HamiltonJacobiBellman.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/HamiltonJacobiBellman.cxx

48.22 Verdandi::IsotropicBalgovindMatrix Class Reference

This class defines a covariance matrix in isotropic Balgovind form.
#include <IsotropicBalgovindMatrix.hxx>

Public Member Functions

- IsotropicBalgovindMatrix (T x_min, T delta_x, int Nx, T y_min, T delta_y, int Ny, T length_scale, T variance)
  Constructor for 2D cases.
- IsotropicBalgovindMatrix (T x_min, T delta_x, int Nx, T y_min, T delta_y, int Ny, T z_min, T delta_z, int Nz, T length_scale, T variance)
  Constructor for 3D cases.
- int GetM () const
  Returns the number of rows.
- int GetN () const
  Returns the number of columns.
- T operator() (int i, int j) const
  Access to one matrix entry.
- void GetRow (int i, Vector<T> &row) const
  Access to a row.
- void GetCol (int i, Vector<T> &column) const
  Access to a column.

Protected Attributes

- Vector<T> min_
  Coordinates of the domain first cell center.
- Vector<T> step_
  Space steps.
- Vector<int> N_
  Number of points along each dimension.
- T length_scale_
  Length scale.
- T variance_
  Variance.
- int dimension_
  Matrix dimension.
48.22.1 Detailed Description

This class defines a covariance matrix in isotropic Balgovind form.

In a covariance matrix in isotropic Balgovind form, the covariance between two points only depends on the Euclidean distances between the points. For example, in the 2D case, if the entry \((i, j)\) of the matrix is the covariance between the values at \((x_i, y_i)\) and \((x_j, y_j)\), its value will be:

\[
B_{i,j} = v \left( 1 + \frac{\sqrt{(x_j-x_i)^2 + (y_j-y_i)^2}}{L} \right) \exp \left( -\frac{\sqrt{(x_j-x_i)^2 + (y_j-y_i)^2}}{L} \right)
\]

where \(v\) is a variance, and \(L\) is the decorrelation length.

48.22.2 Constructor & Destructor Documentation

48.22.2.1 Verdandi::IsotropicBalgovindMatrix::IsotropicBalgovindMatrix ( T x_min, T delta_x, int Nx, T y_min, T delta_y, int Ny, T length_scale, T variance )

Constructor for 2D cases.

This constructors builds an isotropic Balgovind matrix for 2D regular grids.

Parameters

<table>
<thead>
<tr>
<th>x_min</th>
<th>abscissa of the center of the lower-left grid cell.</th>
</tr>
</thead>
<tbody>
<tr>
<td>delta_x</td>
<td>step along x.</td>
</tr>
<tr>
<td>Nx</td>
<td>number of cells along x.</td>
</tr>
<tr>
<td>y_min</td>
<td>ordinate of the center of the lower-left grid cell.</td>
</tr>
<tr>
<td>delta_y</td>
<td>step along y.</td>
</tr>
<tr>
<td>Ny</td>
<td>number of cells along y.</td>
</tr>
<tr>
<td>length_scale</td>
<td>decorrelation length.</td>
</tr>
<tr>
<td>variance</td>
<td>variance.</td>
</tr>
</tbody>
</table>

Definition at line 52 of file IsotropicBalgovindMatrix.cxx.

48.22.2.2 Verdandi::IsotropicBalgovindMatrix::IsotropicBalgovindMatrix ( T x_min, T delta_x, int Nx, T y_min, T delta_y, int Ny, T z_min, T delta_z, int Nz, T length_scale, T variance )

Constructor for 3D cases.

This constructors builds an isotropic Balgovind matrix for 3D regular grids.

Parameters

<table>
<thead>
<tr>
<th>x_min</th>
<th>abscissa of the center of the lower-left grid cell.</th>
</tr>
</thead>
<tbody>
<tr>
<td>delta_x</td>
<td>step along x.</td>
</tr>
<tr>
<td>Nx</td>
<td>number of cells along x.</td>
</tr>
<tr>
<td>y_min</td>
<td>ordinate of the center of the lower-left grid cell.</td>
</tr>
<tr>
<td>delta_y</td>
<td>step along y.</td>
</tr>
<tr>
<td>Ny</td>
<td>number of cells along y.</td>
</tr>
<tr>
<td>z_min</td>
<td>ordinate of the center of the bottom grid cells.</td>
</tr>
<tr>
<td>delta_z</td>
<td>step along z.</td>
</tr>
<tr>
<td>Nz</td>
<td>number of cells along z.</td>
</tr>
<tr>
<td>length_scale</td>
<td>decorrelation length.</td>
</tr>
<tr>
<td>variance</td>
<td>variance.</td>
</tr>
</tbody>
</table>

Definition at line 86 of file IsotropicBalgovindMatrix.cxx.
48.22.3 Member Function Documentation

48.22.3.1 void Verdandi::IsotropicBalgovindMatrix::GetCol ( int j, Vector<T> & column ) const

Access to a column.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>j</th>
<th>column index.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>column</td>
<td>the j-th row.</td>
</tr>
</tbody>
</table>

Definition at line 177 of file IsotropicBalgovindMatrix.cxx.

48.22.3.2 int Verdandi::IsotropicBalgovindMatrix::GetM ( ) const [inline]

Returns the number of rows.

Returns

The number of rows.

Definition at line 117 of file IsotropicBalgovindMatrix.cxx.

48.22.3.3 int Verdandi::IsotropicBalgovindMatrix::GetN ( ) const [inline]

Returns the number of columns.

Returns

The number of columns.

Definition at line 128 of file IsotropicBalgovindMatrix.cxx.

48.22.3.4 void Verdandi::IsotropicBalgovindMatrix::GetRow ( int i, Vector<T> & row ) const

Access to a row.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>i</th>
<th>row index.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>row</td>
<td>the i-th row.</td>
</tr>
</tbody>
</table>

Definition at line 163 of file IsotropicBalgovindMatrix.cxx.

48.22.3.5 T Verdandi::IsotropicBalgovindMatrix::operator() ( int i, int j ) const

Access to one matrix entry.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>i</th>
<th>row index.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>j</td>
<td>column index.</td>
</tr>
</tbody>
</table>
Returns

The value of the entry \((i, j)\) of the matrix.

Definition at line 141 of file IsotropicBalgovindMatrix.hxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/error/IsotropicBalgovindMatrix.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/error/IsotropicBalgovindMatrix.cxx

48.23 Verdandi::LinearObservationManager Class Reference

Linear observation operator.

#include <LinearObservationManager.hxx>

Inheritance diagram for Verdandi::LinearObservationManager:

Verdandi::VerdandiBase

Verdandi::LinearObservationManager

Public Types

- typedef Matrix<T> tangent_linear_operator
  Type of the tangent linear operator.
- typedef Matrix<T> error_variance
  Type of the observation error covariance matrix.
- typedef Vector<T> tangent_linear_operator_row
  Type of a row of the tangent linear operator.
- typedef Vector<T> observation
  Type of the observation vector.
- typedef Vector<T> observation_vector
  Type of the observation vector.
- typedef Vector2<T> observation_vector2
  Type of the observation vector 2.
- typedef Vector3<T> observation_vector3
  Type of the observation vector 3.
- typedef Vector<int> variable_vector
  Type of the variable vector.
- typedef Vector2<int> variable_vector2
  Type of the variable vector 2.
- typedef Vector3<int> variable_vector3
  Type of the variable vector 3.
- typedef Vector<int> index_vector
  Type of the index vector.
- typedef Vector2<int> index_vector2
  Type of the index vector 2.
- typedef Vector3<int> index_vector3
  Type of the index vector 3.
typedef Vector< double > time_vector
Type of the time vector.

typedef Vector2< double > time_vector2
Type of the time vector 2.

typedef Vector3< double > time_vector3
Type of the time vector 3.

Public Member Functions

- LinearObservationManager ()
  Default constructor.

- template<class Model >
  LinearObservationManager (Model &model, string configuration_file)
  Main constructor.

- ~LinearObservationManager ()
  Destructor.

- template<class Model >
  void Initialize (Model &model, string configuration_file)
  Initializes the observation manager.

- template<class Model >
  void InitializeOperator (Model &model, string configuration_file)
  Initializes the observation operator H.

- void DiscardObservation (bool discard_observation)
  Activates or deactivates the option 'discard_observation'.

- int CreateTrack ()
  Creates a new track.

- void SetTrack (int track)
  Sets the track to a given track.

- template<class Model >
  void SetTime (Model &model, double time)
  Sets the time of observations to be loaded.

- void SetTime (double time)
  Sets the time of observations to be loaded.

- void SetAvailableTime (double time, time_vector &available_time)
  Sets available observation times at a given time.

- void SetAvailableTime (double time_inf, double time_sup, time_vector &available_time)
  Sets available observation times at a given time interval.

- void SetAvailableTime (double time, double time_inf, double time_sup, int selection_policy, time_vector &available_time)
  Sets available observation times at a given time interval.

- void GetFlattenedObservation (double time, observation_vector &observation)
  Gets observations flattened over a list of times.

- void GetFlattenedObservation (double time_inf, double time_sup, observation_vector &observation)
  Gets observations flattened over a list of times.

- void GetFlattenedObservation (observation_vector &observation)
  Gets observations flattened over a list of times.

- void GetFlattenedObservation (const time_vector &available_time, observation_vector &observation)
  Gets observations flattened over a list of times.

- void GetFlattenedObservation (double time, variable_vector &observation_variable, observation_vector &observation)
  Gets observations flattened over a list of times.

- void GetFlattenedObservation (double time, variable_vector &observation_variable, observation_vector &observation)
  Gets observations flattened over a list of times.
- void GetFlattenedObservation (double time_inf, double time_sup, variable_vector &observation_variable, observation_vector &observation)

  Gets observations flattened over a list of times.

- void GetFlattenedObservation (variable_vector &observation_variable, observation_vector &observation)

  Gets observations flattened over a list of times.

- void GetFlattenedObservation (const time_vector &available_time, variable_vector &observation_variable, observation_vector &observation)

  Gets observations flattened over a list of times.

- void GetFlattenedObservation (double time_inf, double time_sup, variable_vector &observation_variable, index_vector &observation_index, observation_vector &observation)

  Gets observations flattened over a list of times.

- void GetFlattenedObservation (variable_vector &observation_variable, index_vector &observation_index, observation_vector &observation)

  Gets observations flattened over a list of times.

- void GetFlattenedObservation (const time_vector &available_time, variable_vector &observation_variable, index_vector &observation_index, observation_vector &observation)

  Gets observations flattened over a list of times.

- void GetAggregatedObservation (double time, observation_vector &observation)

  Gets observations aggregated over a list of times.

- void GetAggregatedObservation (double time_inf, double time_sup, observation_vector &observation)

  Gets observations aggregated over a list of times.

- void GetAggregatedObservation (observation_vector &observation)

  Gets observations aggregated over a list of times.

- void GetAggregatedObservation (const time_vector &available_time, observation_vector &observation)

  Gets observations aggregated over a list of times.

- void GetAggregatedObservation (double time, variable_vector &observation_variable, observation_vector2 &observation2)

  Gets observations aggregated over a list of times.

- void GetAggregatedObservation (double time_inf, double time_sup, variable_vector &observation_variable, observation_vector2 &observation2)

  Gets observations aggregated over a list of times.

- void GetAggregatedObservation (variable_vector &observation_variable, observation_vector2 &observation2)

  Gets observations aggregated over a list of times.

- void GetAggregatedObservation (const time_vector &available_time, variable_vector &observation_variable, observation_vector2 &observation2)

  Gets observations aggregated over a list of times.

- void GetAggregatedObservation (double time, variable_vector &observation_variable, index_vector2 &observation_index2, observation_vector2 &observation2)

  Gets observations aggregated over a list of times.

- void GetAggregatedObservation (double time_inf, double time_sup, variable_vector &observation_variable, index_vector2 &observation_index2, observation_vector2 &observation2)

  Gets observations aggregated over a list of times.

- void GetAggregatedObservation (variable_vector &observation_variable, index_vector2 &observation_index2, observation_vector2 &observation2)

  Gets observations aggregated over a list of times.

- void GetAggregatedObservation (const time_vector &available_time, variable_vector &observation_variable, index_vector2 &observation_index2, observation_vector2 &observation2)

  Gets observations aggregated over a list of times.
- void GetRawObservation (double time, observation_vector2 &observation2)
  Gets available observations at a given time.
- void GetRawObservation (double time_inf, double time_sup, observation_vector2 &observation2)
  Gets observations available in a given interval.
- void GetRawObservation (observation_vector2 &observation2)
  Gets observations at the current time.
- void GetRawObservation (const time_vector &available_time, observation_vector2 &observation2)
  Gets observations of a list of times.
- void GetRawObservation (double time, variable_vector2 &observation_variable2, observation_vector3 &observation3)
  Gets available observations at a given time.
- void GetRawObservation (double time_inf, double time_sup, variable_vector2 &observation_variable2, observation_vector3 &observation3)
  Gets observations available in a given interval.
- void GetRawObservation (variable_vector2 &observation_variable2, observation_vector3 &observation3)
  Gets observations at the current time.
- void GetRawObservation (const time_vector &available_time, variable_vector2 &observation_variable2, observation_vector3 &observation3)
  Gets observations of a list of times.
- void GetRawObservation (double time, variable_vector2 &observation_variable2, index_vector3 &observation_index3, observation_vector3 &observation3)
  Gets available observations at a given time.
- void GetRawObservation (double time_inf, double time_sup, variable_vector2 &observation_variable2, index_vector3 &observation_index3, observation_vector3 &observation3)
  Gets observations available in a given interval.
- void GetRawObservation (variable_vector2 &observation_variable2, index_vector3 &observation_index3, observation_vector3 &observation3)
  Gets observations at the current time.
- void GetRawObservation (const time_vector &available_time, variable_vector2 &observation_variable2, index_vector3 &observation_index3, observation_vector3 &observation3)
  Gets observations of a list of times.
- void ReadObservationVariable (const time_vector &available_time, variable_vector2 &observation_variable2) const
  Builds variables vector associated with given observations.
- void ReadObservation (const time_vector &available_time, const variable_vector2 &observation_variable2, observation_vector3 &observation3) const
  Builds observations associated with given times and variables.
- void ReadObservation (const time_vector &available_time, observation_vector2 &observation2) const
  Builds observations associated with given times.
- void ReadObservation (ifstream &file_stream, double time, int variable, observation_vector &observation) const
  Reads observation from observation file given a time and a variable.
- void ReadObservationIndex (const time_vector &available_time, const variable_vector2 &observation_variable2, index_vector3 &observation_index3) const
  Reads observations indexes.
- void ReadObservationTriangleWidth (double time_inf, double time_sup, Vector< double > &width_left, Vector< double > &width_right) const
  Reads triangle width associated with observations of a given interval.
- observation & GetObservation ()
  Gets observation.
- template<class state>
  observation & GetInnovation (const state &x)
Gets innovation.

- `bool HasObservation() const`
  Indicates if some observations are available at current time.

- `int GetNobservation() const`
  Gets Nobservation value.

- `bool IsOperatorSparse() const`
  Checks whether the observation operator is available in a sparse matrix.

- `bool IsErrorSparse() const`
  Checks whether the observation error covariance matrix is sparse.

- `bool HasErrorMatrix() const`
  Checks whether the observation error covariance is available in a matrix.

- `template<class state, class mat>
  void GetNudgingMatrix(const state &x, mat &M) const`
  Returns the nudging matrix.

- `template<class state>
  void ApplyOperator(const state &x, observation &y) const`
  Applies the operator to a given vector.

- `template<class state>
  void ApplyTangentLinearOperator(const state &x, observation &y) const`
  Applies the tangent linear operator to a given vector.

- `T GetTangentLinearOperator(int i, int j) const`
  Linearized observation operator.

- `tangent_linear_operator_row & GetTangentLinearOperatorRow(int row)`
  Linearized observation operator.

- `template<class state>
  void ApplyAdjointOperator(const state &x, observation &y) const`
  Applies the adjoint operator to a given vector.

- `bool HasBLUECorrection() const`
  Checks whether a BLUE correction is available.

- `void GetBLUECorrection(Vector<T> &BLUE_correction) const`
  Gets the BLUE correction.

- `T GetErrorVariance(int i, int j) const`
  Observation error covariance.

- `const error_variance & GetErrorVariance() const`
  Observation error covariance matrix.

- `const error_variance & GetErrorVarianceInverse() const`
  Inverse of the observation error covariance matrix.

- `string GetName() const`
  Returns the name of the class.

- `void Message(string message)`
  Receives and handles a message.

- `template<class state, class mat>
  void GetNudgingMatrix(const state &x, mat &M) const`
  Returns the nudging matrix.

**Static Public Member Functions**

- `static void StaticMessage(void *object, string message)`
  Receives and handles a message with a static method.
Protected Attributes

- string observation_file_
  File that stores the observations.
- string observation_file_type_
  Type of the file.
- string observation_dataset_path_
  Path to the dataset where observations are stored (HDF5 filetype).
- string observation_type_
  How are stored the observations.
- int Nobservation_
  Total number of observations at current time.
- size_t Nbyte_observation_
  Size in bytes of an observation vector.
- bool is_delta_t_constant_
  Is the time interval between two observations constant?
- double Delta_t_
  Period with which observations are available (if constant).
- Vector<double> observation_time_
- string observation_time_file_
- int Nskip_
  Period with which available observations are actually loaded.
- double initial_time_
  First time at which observations are available.
- double final_time_
  Final time at which observations are available.
- double time_
  Requested time.
- time_vector available_time_
  Available observation time of the time interval.
- Vector<double> contribution_
  Contribution associated with available observations.
- OBSERVATION_AGGREGATOR<T> observation_aggregator_
  Observations aggregator.
- tangent_linear_operator tangent_operator_matrix_
  Tangent operator matrix (H).
- bool operator_scaled_identity_
  Is the operator a scaled identity matrix?
- T operator_diagonal_value_
  In case of a scaled identity operator.
- T error_variance_value_
  Observation error variance.
- error_variance error_variance_
  Observation error covariance matrix (R).
- error_variance error_variance_inverse_
  Inverse of the observation error covariance matrix (R).
- string width_file_
  File that stores the observations.
- int Nstate_model_
  The size of a model state.
-observation observation_
48.23.1 Detailed Description

Linear observation operator.

Template Parameters

\[ T \] the type of floating-point numbers.

48.23.2 Constructor & Destructor Documentation

48.23.2.1 Verdandi::LinearObservationManager::LinearObservationManager ( )

Default constructor.

It entirely defines the operator: no dimension or size is associated with this implementation.

Definition at line 45 of file LinearObservationManager.cxx.

48.23.2.2 \texttt{template< class Model >} Verdandi::LinearObservationManager::LinearObservationManager ( Model & model, string configuration_file )

Main constructor.

Parameters

| in   | model | model. |
| in   | configuration_file | configuration_file. |

Template Parameters

\[ Model \] the model type; e.g. ShallowWater<\texttt{double}>

Definition at line 59 of file LinearObservationManager.cxx.

48.23.3 Member Function Documentation

48.23.3.1 \texttt{template< class state >} void Verdandi::LinearObservationManager::ApplyAdjointOperator ( const state & x, observation & y ) const

Applies the adjoint operator to a given vector.

Parameters

| in   | x    | a vector. |
| out  | y    | the value of the operator at x. |

Definition at line 2060 of file LinearObservationManager.cxx.
48.23.3.2 template<class state> void Verdandi::LinearObservationManager::ApplyOperator ( const state & x, observation & y ) const

Applies the operator to a given vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x</th>
<th>a vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>y</td>
<td>the value of the operator at x.</td>
</tr>
</tbody>
</table>

Definition at line 1958 of file LinearObservationManager.cxx.

48.23.3.3 template<class state> void Verdandi::LinearObservationManager::ApplyTangentLinearOperator ( const state & x, observation & y ) const

Applies the tangent linear operator to a given vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x</th>
<th>a vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>y</td>
<td>the value of the tangent linear operator at x.</td>
</tr>
</tbody>
</table>

Definition at line 1984 of file LinearObservationManager.cxx.

48.23.3.4 int Verdandi::LinearObservationManager::CreateTrack ( )

Creates a new track.

Returns

The index of the new track.

Definition at line 334 of file LinearObservationManager.cxx.

48.23.3.5 void Verdandi::LinearObservationManager::DiscardObservation ( bool discard_observation )

Activates or deactivates the option 'discard_observation'.

Parameters

| in     | discard_observation | if set to true, each observation will be used at most one time. |

Definition at line 322 of file LinearObservationManager.cxx.

48.23.3.6 void Verdandi::LinearObservationManager::GetAggregatedObservation ( double time, observation_vector & observation )

Gets observations aggregated over a list of times.

The observations available at the given time are loaded and aggregated.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>the given time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation</td>
<td>the aggregated observations.</td>
</tr>
</tbody>
</table>

Definition at line 322 of file LinearObservationManager.cxx.
48.23.3.7 void Verdandi::LinearObservationManager::GetAggregatedObservation ( double time_inf, double time_sup, observation_vector & observation )

Gets observations aggregated over a list of times.
The observations in the interval [time_inf, time_sup] are loaded and aggregated.

Parameters

| in         | time_inf | lower bound of the given interval. |
| in         | time_sup | upper bound (excluded) of the given interval. |
| out        | observation | the aggregated observations. |

48.23.3.8 void Verdandi::LinearObservationManager::GetAggregatedObservation ( observation_vector & observation )

Gets observations aggregated over a list of times.
The observations available are loaded and aggregated.

Parameters

| out | observation | the aggregated observations. |

48.23.3.9 void Verdandi::LinearObservationManager::GetAggregatedObservation ( const time_vector & available_time, observation_vector & observation )

Gets observations aggregated over a list of times.
The observations available at the times available_time are loaded and aggregated.

Parameters

| in         | available_time | the given observation time vector. |
| out        | observation | the aggregated observations. |

48.23.3.10 void Verdandi::LinearObservationManager::GetAggregatedObservation ( double time, variable_vector & observation_variable, observation_vector2 & observation2 )

Gets observations aggregated over a list of times.
The observations available at the given time are loaded and aggregated.

Parameters

| in         | time | the given time. |
| out        | observation_variable | variables associated with the observations. |
| out        | observation2 | the aggregated observations. |

Definitions at line 878, 897, 914, 928 of file LinearObservationManager.cxx.
48.23.3.11 void Verdandi::LinearObservationManager::GetAggregatedObservation ( double time_inf, double time_sup, variable_vector & observation_variable, observation_vector2 & observation2 )

Gets observations aggregated over a list of times.
The observations in the interval \([\text{time} \_\text{inf}, \text{time} \_\text{sup}]\) are loaded and aggregated.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>time_inf</td>
<td>lower bound of the given interval.</td>
</tr>
<tr>
<td>in</td>
<td>time_sup</td>
<td>upper bound (excluded) of the given interval.</td>
</tr>
<tr>
<td>out</td>
<td>observation__variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation2</td>
<td>the aggregated observations.</td>
</tr>
</tbody>
</table>

Definition at line 974 of file LinearObservationManager.cxx.

48.23.3.12 void Verdandi::LinearObservationManager::GetAggregatedObservation ( variable_vector & observation_variable, observation_vector2 & observation2 )

Gets observations aggregated over a list of times.
The observations available are loaded and aggregated.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation__variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation2</td>
<td>the aggregated observations.</td>
</tr>
</tbody>
</table>

Definition at line 995 of file LinearObservationManager.cxx.

48.23.3.13 void Verdandi::LinearObservationManager::GetAggregatedObservation ( const time_vector & available_time, variable_vector & observation_variable, observation_vector2 & observation2 )

Gets observations aggregated over a list of times.
The observations available at the times \(\text{available\_time}\) are loaded and aggregated.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>available_time</td>
<td>the given observation time vector.</td>
</tr>
<tr>
<td>out</td>
<td>observation__variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation2</td>
<td>the aggregated observations.</td>
</tr>
</tbody>
</table>

Definition at line 1013 of file LinearObservationManager.cxx.

48.23.3.14 void Verdandi::LinearObservationManager::GetAggregatedObservation ( double time, variable_vector & observation_variable, index_vector2 & observation\_index2, observation_vector2 & observation2 )

Gets observations aggregated over a list of times.
The observations available at the given time are loaded and aggregated.
Parameters  

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>the given time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation_variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation_index2</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation2</td>
<td>the aggregated observations.</td>
</tr>
</tbody>
</table>

Definition at line 1045 of file LinearObservationManager.cxx.

48.23.3.15 void Verdandi::LinearObservationManager::GetAggregatedObservation ( double time\_inf, double time\_sup, variable\_vector & observation\_variable, index\_vector2 & observation\_index2, observation\_vector2 & observation2 )

Gets observations aggregated over a list of times.

The observations in the interval \([\text{time\_inf}, \text{time\_sup}]\) are loaded and aggregated.

Parameters  

<table>
<thead>
<tr>
<th>in</th>
<th>time_inf</th>
<th>lower bound of the given interval.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>time_sup</td>
<td>upper bound (excluded) of the given interval.</td>
</tr>
<tr>
<td>out</td>
<td>observation_variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation_index2</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation2</td>
<td>the aggregated observations.</td>
</tr>
</tbody>
</table>

Definition at line 1070 of file LinearObservationManager.cxx.

48.23.3.16 void Verdandi::LinearObservationManager::GetAggregatedObservation ( variable\_vector & observation\_variable, index\_vector2 & observation\_index2, observation\_vector2 & observation2 )

Gets observations aggregated over a list of times.

The observations available are loaded and aggregated.

Parameters  

<table>
<thead>
<tr>
<th>out</th>
<th>observation_variable</th>
<th>variables associated with the observations.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation_index2</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation2</td>
<td>the aggregated observations.</td>
</tr>
</tbody>
</table>

Definition at line 1093 of file LinearObservationManager.cxx.

48.23.3.17 void Verdandi::LinearObservationManager::GetAggregatedObservation ( const time\_vector & available\_time, variable\_vector & observation\_variable, index\_vector2 & observation\_index2, observation\_vector2 & observation2 )

Gets observations aggregated over a list of times.

The observations available at the times \(\text{available\_time}\) are loaded and aggregated.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>available_time</th>
<th>the given observation time vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation_variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation_index2</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation2</td>
<td>the aggregated observations.</td>
</tr>
</tbody>
</table>

Definition at line 1114 of file LinearObservationManager.cxx.

48.23.3.18 void Verdandi::LinearObservationManager::GetBLUECorrection ( Vector<T> & BLUE_correction ) const

 Gets the BLUE correction.

Parameters

| out | BLUE_correction | BLUE correction vector. |

Definition at line 2092 of file LinearObservationManager.cxx.

48.23.3.19 T Verdandi::LinearObservationManager::GetErrorVariance ( int i, int j ) const

Observation error covariance.

Parameters

| in | i | row index. |
|    | j | column index. |

Returns

The element \((i, j)\) of the observation error covariance.

Definition at line 2106 of file LinearObservationManager.cxx.

48.23.3.20 const LinearObservationManager<T> & Verdandi::LinearObservationManager::GetErrorVariance ( ) const

Observation error covariance matrix.

Returns

The matrix of the observation error covariance.

Definition at line 2121 of file LinearObservationManager.cxx.

48.23.3.21 const LinearObservationManager<T> & Verdandi::LinearObservationManager::GetErrorVarianceInverse ( ) const

Inverse of the observation error covariance matrix.

Returns

Inverse of the matrix of the observation error covariance.

Definition at line 2133 of file LinearObservationManager.cxx.
48.23.3.22 void Verdandi::LinearObservationManager::GetFlattenedObservation ( double time, observation_vector & observation )

Gets observations flattened over a list of times.
The observations available at time `time` are loaded and concatenated in a vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>the given time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation</td>
<td>the observation to be flattened.</td>
</tr>
</tbody>
</table>

Definition at line 625 of file LinearObservationManager.cxx.

48.23.3.23 void Verdandi::LinearObservationManager::GetFlattenedObservation ( double time_inf, double time_sup, observation_vector & observation )

Gets observations flattened over a list of times.
The observations in the interval `[time_inf, time_sup]` are loaded and concatenated in a vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time_inf</th>
<th>lower bound of the given interval.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>time_sup</td>
<td>upper bound (excluded) of the given interval.</td>
</tr>
<tr>
<td>out</td>
<td>observation</td>
<td>the observation to be flattened.</td>
</tr>
</tbody>
</table>

Definition at line 643 of file LinearObservationManager.cxx.

48.23.3.24 void Verdandi::LinearObservationManager::GetFlattenedObservation ( observation_vector & observation )

Gets observations flattened over a list of times.
The observations available are loaded and concatenated in a vector.

Parameters

| out | observation | the observation to be loaded. |

Definition at line 658 of file LinearObservationManager.cxx.

48.23.3.25 void Verdandi::LinearObservationManager::GetFlattenedObservation ( const time_vector & available_time, observation_vector & observation )

Gets observations flattened over a list of times.
The observations available at the times `available_time` are loaded and concatenated in a vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>available_time</th>
<th>the given observation time vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 672 of file LinearObservationManager.cxx.
48.23.3.26 void Verdandi::LinearObservationManager::GetFlattenedObservation ( double \textit{time}, \texttt{variable\_vector \& \textit{observation\_variable}}, \texttt{observation\_vector \& \textit{observation}} )

Gets observations flattened over a list of times.

The observations available at time \textit{time} are loaded and concatenated in a vector.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{time}</td>
</tr>
<tr>
<td>\texttt{observation_variable}</td>
</tr>
<tr>
<td>\textit{observation}</td>
</tr>
</tbody>
</table>

Definition at line 694 of file LinearObservationManager.cxx.

48.23.3.27 void Verdandi::LinearObservationManager::GetFlattenedObservation ( double \textit{time\_inf}, double \textit{time\_sup}, \texttt{variable\_vector \& \textit{observation\_variable}}, \texttt{observation\_vector \& \textit{observation}} )

Gets observations flattened over a list of times.

The observations in the interval \([\textit{time\_inf}, \textit{time\_sup}]\) are loaded and concatenated in a vector.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{time_inf}</td>
</tr>
<tr>
<td>\textit{time_sup}</td>
</tr>
<tr>
<td>\texttt{observation_variable}</td>
</tr>
<tr>
<td>\textit{observation}</td>
</tr>
</tbody>
</table>

Definition at line 716 of file LinearObservationManager.cxx.

48.23.3.28 void Verdandi::LinearObservationManager::GetFlattenedObservation ( \texttt{variable\_vector \& \textit{observation\_variable}}, \texttt{observation\_vector \& \textit{observation}} )

Gets observations flattened over a list of times.

The observations available are loaded and concatenated in a vector.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{observation_variable}</td>
</tr>
<tr>
<td>\textit{observation}</td>
</tr>
</tbody>
</table>

Definition at line 735 of file LinearObservationManager.cxx.

48.23.3.29 void Verdandi::LinearObservationManager::GetFlattenedObservation ( const \texttt{time\_vector \& \textit{available\_time}}, \texttt{variable\_vector \& \textit{observation\_variable}}, \texttt{observation\_vector \& \textit{observation}} )

Gets observations flattened over a list of times.

The observations available at the times \textit{available\_time} are loaded and concatenated in a vector.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{available_time}</td>
</tr>
<tr>
<td>\texttt{observation_variable}</td>
</tr>
<tr>
<td>\textit{observation}</td>
</tr>
</tbody>
</table>

Definition at line 754 of file LinearObservationManager.cxx.
300 Class Documentation

out  observation  the observation to be loaded.

Definition at line 753 of file LinearObservationManager.cxx.

48.23.3.30 void Verdandi::LinearObservationManager::GetFlattenedObservation ( double time, variable_vector & observation_variable, index_vector & observation_index, observation_vector & observation )

Gets observations flattened over a list of times.
The observations available at time time are loaded and concatenated in a vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>the given time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation_variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation_index</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation</td>
<td>the observation to be flattened.</td>
</tr>
</tbody>
</table>

Definition at line 780 of file LinearObservationManager.cxx.

48.23.3.31 void Verdandi::LinearObservationManager::GetFlattenedObservation ( double time_inf, double time_sup, variable_vector & observation_variable, index_vector & observation_index, observation_vector & observation )

Gets observations flattened over a list of times.
The observations in the interval [time_inf, time_sup] are loaded and concatenated in a vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time_inf</th>
<th>lower bound of the given interval.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>time_sup</td>
<td>upper bound (excluded) of the given interval.</td>
</tr>
<tr>
<td>out</td>
<td>observation_variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation_index</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation</td>
<td>the observation to be flattened.</td>
</tr>
</tbody>
</table>

Definition at line 804 of file LinearObservationManager.cxx.

48.23.3.32 void Verdandi::LinearObservationManager::GetFlattenedObservation ( variable_vector & observation_variable, index_vector & observation_index, observation_vector & observation )

Gets observations flattened over a list of times.
The observations available are loaded and concatenated in a vector.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>observation_variable</th>
<th>variables associated with the observations.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation_index</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>
void Verdandi::LinearObservationManager::GetFlattenedObservation ( const time_vector& available_time, variable_vector& observation_variable, index_vector& observation_index, observation_vector& observation )

Gets observations flattened over a list of times.
The observations available at the times available_time are loaded and concatenated in a vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>available_time</th>
<th>the given observation time vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation_variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation_index</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 845 of file LinearObservationManager.cxx.

template<class state > LinearObservationManager< T >::observation & Verdandi::LinearObservationManager::GetInnovation ( const state & x )

Gets innovation.

Parameters

| in | state | state vector. |

Returns
The innovation vector.

Definition at line 1843 of file LinearObservationManager.cxx.

string Verdandi::LinearObservationManager::GetName ( ) const [virtual]

Returns the name of the class.

Returns
The name of the class.

Reimplemented from Verdandi::VerdandiBase.

Definition at line 2146 of file LinearObservationManager.cxx.

int Verdandi::LinearObservationManager::GetNobservation ( ) const

Gets Nobservation_value.

Returns
The total number of observation at current time.

Definition at line 1872 of file LinearObservationManager.cxx.
template<class state, class mat> void Verdandi::LinearObservationManager::GetNudgingMatrix ( const & x, mat & M ) const

Returns the nudging matrix.

<table>
<thead>
<tr>
<th>in</th>
<th>x</th>
<th>a vector representing the state.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>output</td>
<td>the nudging matrix at x.</td>
</tr>
</tbody>
</table>

Definition at line 192 of file ObservationManagerTemplate.hxx.

template<class state, class mat> void Verdandi::LinearObservationManager::GetNudgingMatrix ( const state & x, mat & M ) const

Returns the nudging matrix.

<table>
<thead>
<tr>
<th>in</th>
<th>x</th>
<th>a vector representing the state.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>output</td>
<td>the nudging matrix at x.</td>
</tr>
</tbody>
</table>

Definition at line 1937 of file LinearObservationManager.hxx.

LinearObservationManager<T>::observation & Verdandi::LinearObservationManager::GetObservation ( )

Gets observation.

Returns

The observation vector.

Definition at line 1823 of file LinearObservationManager.hxx.

void Verdandi::LinearObservationManager::GetRawObservation ( double time, observation_vector2 & observation2 )

Gets available observations at a given time.

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>the given time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation2</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 1152 of file LinearObservationManager.hxx.

void Verdandi::LinearObservationManager::GetRawObservation ( double time_inf, double time_sup, observation_vector2 & observation2 )

Gets observations available in a given interval.

<table>
<thead>
<tr>
<th>in</th>
<th>time_inf</th>
<th>lower bound of the given interval.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>time_sup</td>
<td>upper bound (excluded) of the given interval.</td>
</tr>
<tr>
<td>out</td>
<td>observation2</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>
48.23.3.42 void Verdandi::LinearObservationManager::GetRawObservation ( observation_vector2 & observation2 )

Gets observations at the current time.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>out</strong></td>
<td>observation2</td>
</tr>
</tbody>
</table>

Definition at line 1186 of file LinearObservationManager.cxx.

48.23.3.43 void Verdandi::LinearObservationManager::GetRawObservation ( const time_vector & available_time, observation_vector2 & observation2 )

Gets observations of a list of times.

Parameters

<table>
<thead>
<tr>
<th><strong>in</strong></th>
<th>available_time</th>
<th>the given observation time vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>out</strong></td>
<td>observation2</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 1199 of file LinearObservationManager.cxx.

48.23.3.44 void Verdandi::LinearObservationManager::GetRawObservation ( double time, variable_vector2 & observation_variable2, observation_vector3 & observation3 )

Gets available observations at a given time.

Parameters

<table>
<thead>
<tr>
<th><strong>in</strong></th>
<th>time</th>
<th>the given time.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>out</strong></td>
<td>observation_variable2</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td><strong>out</strong></td>
<td>observation3</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 1219 of file LinearObservationManager.cxx.

48.23.3.45 void Verdandi::LinearObservationManager::GetRawObservation ( double time_inf, double time_sup, variable_vector2 & observation_variable2, observation_vector3 & observation3 )

Gets observations available in a given interval.

Parameters

<table>
<thead>
<tr>
<th><strong>in</strong></th>
<th>time_inf</th>
<th>lower bound of the given interval.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td>time_sup</td>
<td>upper bound (excluded) of the given interval.</td>
</tr>
<tr>
<td><strong>out</strong></td>
<td>observation_variable2</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td><strong>out</strong></td>
<td>observation3</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 1241 of file LinearObservationManager.cxx.
48.23.3.46  void Verdandi::LinearObservationManager::GetRawObservation ( variable_vector2 & observation_variable2, observation_vector3 & observation3 )

Gets observations at the current time.

Parameters
| out | observation_variable2 | variables associated with the observations. |
| out | observation3           | the observation to be loaded.                |

Definition at line 1262 of file LinearObservationManager.cxx.

48.23.3.47  void Verdandi::LinearObservationManager::GetRawObservation ( const time_vector & available_time, variable_vector2 & observation_variable2, observation_vector3 & observation3 )

Gets observations of a list of times.

Parameters
| in  | available_time        | the given observation time vector.        |
| out | observation_variable2 | variables associated with the observations. |
| out | observation3          | the observation to be loaded.             |

Definition at line 1280 of file LinearObservationManager.cxx.

48.23.3.48  void Verdandi::LinearObservationManager::GetRawObservation ( double time, variable_vector2 & observation_variable2, index_vector3 & observation_index3, observation_vector3 & observation3 )

Gets available observations at a given time.

Parameters
| in  | time          | the given time. |
| out | observation_variable2 | variables associated with the observations. |
| out | observation_index3 | indexes associated with the observations. |
| out | observation3   | the observation to be loaded.               |

Definition at line 1303 of file LinearObservationManager.cxx.

48.23.3.49  void Verdandi::LinearObservationManager::GetRawObservation ( double time_inf, double time_sup, variable_vector2 & observation_variable2, index_vector3 & observation_index3, observation_vector3 & observation3 )

Gets observations available in a given interval.

Parameters
| in  | time_inf       | lower bound of the given interval. |
| in  | time_sup       | upper bound (excluded) of the given interval. |
| out | observation_variable2 | variables associated with the observations. |
| out | observation_index3 | indexes associated with the observations. |
| out | observation3   | the observation to be loaded.               |
48.23 Verdandi::LinearObservationManager Class Reference

Definition at line 1327 of file LinearObservationManager.cxx.

48.23.3.50 void Verdandi::LinearObservationManager::GetRawObservation ( variable_vector2 & observation_variable2, index_vector3 & observation_index3, observation_vector3 & observation3 )

Gets observations at the current time.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>observation_variable2</th>
<th>variables associated with the observations.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation_index3</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation3</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 1350 of file LinearObservationManager.cxx.

48.23.3.51 void Verdandi::LinearObservationManager::GetRawObservation ( const time_vector & available_time, variable_vector2 & observation_variable2, index_vector3 & observation_index3, observation_vector3 & observation3 )

Gets observations of a list of times.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>available_time</th>
<th>the given observation time vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation_variable2</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation_index3</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation3</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 1370 of file LinearObservationManager.cxx.

48.23.3.52 T Verdandi::LinearObservationManager::GetTangentLinearOperator ( int i, int j ) const

Linearized observation operator.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>i</th>
<th>row index.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>j</td>
<td>column index.</td>
</tr>
</tbody>
</table>

Returns

The element \((i, j)\) of the linearized operator.

Definition at line 2000 of file LinearObservationManager.cxx.

48.23.3.53 const LinearObservationManager< T >::tangent_linear_operator& Verdandi::LinearObservationManager::GetTangentLinearOperator ( ) const

Linearized observation operator.
Returns

The matrix of the linearized operator.

Definition at line 2046 of file LinearObservationManager.cxx.

48.23.3.54 LinearObservationManager & Verdandi::LinearObservationManager::GetTangentLinearOperatorRow ( int row )

Linearized observation operator.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>row</th>
<th>row index.</th>
</tr>
</thead>
</table>

Returns

The row row of the linearized operator.

Definition at line 2019 of file LinearObservationManager.cxx.

48.23.3.55 bool Verdandi::LinearObservationManager::HasBLUECorrection ( ) const

Checks whether a BLUE correction is available.

Returns

True if a BLUE correction is available, false otherwise.

Definition at line 2079 of file LinearObservationManager.cxx.

48.23.3.56 bool Verdandi::LinearObservationManager::HasErrorMatrix ( ) const

Checks whether the observation error covariance is available in a matrix.

Returns

True if the observation error covariance is available in a matrix, false otherwise.

Definition at line 1919 of file LinearObservationManager.cxx.

48.23.3.57 template < class Model > void Verdandi::LinearObservationManager::Initialize ( Model & model, string configuration_file )

Initializes the observation manager.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>model</th>
<th>model.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>configuration_file</td>
<td>configuration file.</td>
</tr>
</tbody>
</table>

Template Parameters

| Model | the model type; e.g. ShallowWater<double> |

Definition at line 88 of file LinearObservationManager.cxx.
48.23.3.58 template < class Model > void Verdandi::LinearObservationManager::InitializeOperator ( Model & model, string configuration_file )

Initializes the observation operator $H$.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>model</th>
<th>model.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>configuration_file</td>
<td>configuration file.</td>
</tr>
</tbody>
</table>

Template Parameters

| Model | the model type; e.g. ShallowWater< double > |

Definition at line 191 of file LinearObservationManager.cxx.

48.23.3.59 bool Verdandi::LinearObservationManager::IsErrorSparse ( ) const

Checks whether the observation error covariance matrix is sparse.

Returns

True if the observation error covariance matrix is sparse, false otherwise.

Definition at line 1902 of file LinearObservationManager.cxx.

48.23.3.60 bool Verdandi::LinearObservationManager::IsOperatorSparse ( ) const

Checks whether the observation operator is available in a sparse matrix.

Returns

True if the observation operator is available in a sparse matrix, false otherwise.

Definition at line 1886 of file LinearObservationManager.cxx.

48.23.3.61 void Verdandi::LinearObservationManager::ReadObservation ( const time_vector & available_time, const variable_vector2 & observation_variable2, observation_vector3 & observation3 ) const

Builds observations associated with given times and variables.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>available_time</th>
<th>the given observation time vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>observation_variable2</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation3</td>
<td>the observations.</td>
</tr>
</tbody>
</table>

Definition at line 1417 of file LinearObservationManager.cxx.

48.23.3.62 void Verdandi::LinearObservationManager::ReadObservation ( const time_vector & available_time, observation_vector2 & observation2 ) const

Builds observations associated with given times.
Parameters

| in       | available_time | the given observation time vector. |
| out      | observation2   | the observations.                  |

Definition at line 1459 of file LinearObservationManager.cxx.

48.23.3.63 void Verdandi::LinearObservationManager::ReadObservation ( ifstream & file_stream, double time, int variable, observation_vector & observation ) const

Reads observation from observation file given a time and a variable.

Parameters

| in       | file_stream | the observation file stream. |
| in       | time        | the time.                    |
| in       | variable    | the variable.                |
| out      | observation | the observations.            |

Definition at line 1500 of file LinearObservationManager.cxx.

48.23.3.64 void Verdandi::LinearObservationManager::ReadObservationIndex ( const time_vector & available_time, const variable_vector2 & observation_variable2, index_vector3 & observation_index3 ) const

Reads observations indexes.

Parameters

| in       | available_time | the available time. |
| in       | observation_ - | variable associated with the observations. |
| in       | variable2      |                       |
| out      | observation_ - | the indexes associated with the observations. |
|          | index3         |                       |

Definition at line 1732 of file LinearObservationManager.cxx.

48.23.3.65 void Verdandi::LinearObservationManager::ReadObservationTriangleWidth ( double time_inf, double time_sup, Vector< double > & width_left, Vector< double > & width_right ) const

Reads triangle width associated with observations of a given interval.

Parameters

| in       | time_inf | lower bound of a given interval. |
| in       | time_sup | upper bound (excluded) of a given interval. |
| out      | width_left | left widths associated with observations. |
| out      | width_right | right widths associated with observations. |

Definition at line 1763 of file LinearObservationManager.cxx.

48.23.3.66 void Verdandi::LinearObservationManager::ReadObservationVariable ( const time_vector & available_time, variable_vector2 & observation_variable2 ) const

Builds variables vector associated with given observations.
48.23 Verdandi::LinearObservationManager Class Reference

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>available_time</th>
<th>the given observation time vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation_variable2</td>
<td>variables associated with the observations.</td>
</tr>
</tbody>
</table>

Definition at line 1396 of file LinearObservationManager.cxx.

48.23.3.67 void Verdandi::LinearObservationManager::SetAvailableTime ( double time, time_vector & available_time )

Sets the available observation times at a given time.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>the given time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>available_time</td>
<td>the available observation times.</td>
</tr>
</tbody>
</table>

Definition at line 389 of file LinearObservationManager.cxx.

48.23.3.68 void Verdandi::LinearObservationManager::SetAvailableTime ( double time_inf, double time_sup, time_vector & available_time )

Sets available observation times at a given time interval.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time_inf</th>
<th>lower bound of the given time interval.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>time_sup</td>
<td>upper bound (excluded) of the given time interval.</td>
</tr>
<tr>
<td>in</td>
<td>selection_policy</td>
<td>interval selection policy.</td>
</tr>
<tr>
<td>out</td>
<td>available_time</td>
<td>the available observation times.</td>
</tr>
</tbody>
</table>

Definition at line 415 of file LinearObservationManager.cxx.

48.23.3.69 void Verdandi::LinearObservationManager::SetAvailableTime ( double time, double time_inf, double time_sup, int selection_policy, time_vector & available_time )

Sets available observation times at a given time interval.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time_inf</th>
<th>lower bound of the given time interval.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>time_sup</td>
<td>upper bound (excluded) of the given time interval.</td>
</tr>
<tr>
<td>in</td>
<td>selection_policy</td>
<td>interval selection policy.</td>
</tr>
<tr>
<td>out</td>
<td>available_time</td>
<td>the available observation times.</td>
</tr>
</tbody>
</table>

Definition at line 450 of file LinearObservationManager.cxx.

48.23.3.70 template<class Model > void Verdandi::LinearObservationManager::SetTime ( Model & model, double time )

Sets the time of observations to be loaded.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>model</th>
<th>the model.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>time</td>
<td>a given time.</td>
</tr>
</tbody>
</table>
void Verdandi::LinearObservationManager::SetTime ( double time )

Sets the time of observations to be loaded.

Parameters

| in | time | a given time |

void Verdandi::LinearObservationManager::SetTrack ( int track )

Sets the track to a given track.

Parameters

| in | track | the given track |

Vector< double > Verdandi::LinearObservationManager::observation_time_ [protected]

Times at which observations are available (if the time interval between two observations is not constant).

string Verdandi::LinearObservationManager::observation_time_file_ [protected]

Path to the observation times (needed if the time interval between two observations is not constant).

This class is a linear observation manager written in Python.

Public Member Functions

- def __init__
  
  Initializes the model.

- def SetTime
Sets the current time.

- def HasObservation
  Indicates if some observations are available at a given time.

- def GetObservation
  Returns the number of available observations.

- def ReadObservation
  Returns the observations.

- def ApplyOperator
  Applies the operator to a given vector.

- def ApplyTangentLinearOperator
  Applies the tangent linear operator to a given vector.

- def GetInnovation
  Gets innovation.

- def GetTangentLinearOperator

- def GetTangentLinearOperatorRow
  Returns a row of the tangent linear operator.

- def ApplyAdjointOperator
  Applies the adjoint operator to a given vector.

- def GetErrorVariance
  Returns an element of the observation error covariance matrix, or the full matrix.

- def GetErrorVarianceInverse
  Returns the inverse of the observation error covariance matrix.

- def GetName
  Returns the name of the class.

- def Message
  Receives and handles a message.

Public Attributes

- Nstate_model_
- observation_file_
- observation_type_
- Delta_t_
- Nskip_
- initial_time_
- final_time_
- time_
- width_file_
- Nobservation_
- tangent_operator_matrix_
- error_variance_
- error_variance_inverse_
- Nbyte_observation_

48.24.1 Detailed Description

This class is a linear observation manager written in Python.
48.24.2 Constructor & Destructor Documentation

48.24.2.1 def LinearObservationManager::LinearObservationManager::__init__ ( self, path, Nstate, Nmodel )

Initializes the model.

Parameters

| in | path | The path to the python file |

Definition at line 29 of file LinearObservationManager.py.

48.24.3 Member Function Documentation

48.24.3.1 def LinearObservationManager::LinearObservationManager::ApplyAdjointOperator ( self, state )

Applies the adjoint operator to a given vector.

Parameters

| in | x | a vector |

Returns

the value of the operator at x.

Definition at line 195 of file LinearObservationManager.py.

48.24.3.2 def LinearObservationManager::LinearObservationManager::ApplyOperator ( self, state, observation = None )

Applies the operator to a given vector.

Parameters

| in | x | a vector. |
| out | y | the value of the operator at x. |

Definition at line 142 of file LinearObservationManager.py.

48.24.3.3 def LinearObservationManager::LinearObservationManager::ApplyTangentLinearOperator ( self, state )

Applies the tangent linear operator to a given vector.

This method is called after 'SetTime' set the time at which the observations are requested.

Parameters

| in | x | a vector. |

Returns

the value of the tangent linear operator at x.

Definition at line 155 of file LinearObservationManager.py.
48.24.3.4 def LinearObservationManager::LinearObservationManager::GetErrorVariance ( self, i=None, j=None )

Returns an element of the observation error covariance matrix, or the full matrix.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>i</th>
<th>row index.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>j</td>
<td>column index.</td>
</tr>
</tbody>
</table>

Returns

The element \((i, j)\) of the observation error covariance matrix, or the full matrix.

Definition at line 206 of file LinearObservationManager.py.

48.24.3.5 def LinearObservationManager::LinearObservationManager::GetErrorVarianceInverse ( self )

Returns the inverse of the observation error covariance matrix.

Returns

The inverse of the observation error covariance matrix.

Definition at line 218 of file LinearObservationManager.py.

48.24.3.6 def LinearObservationManager::LinearObservationManager::GetInnovation ( self, state )

Gets innovation.

Parameters

| in     | state | state vector. |

Returns

innovation innovation vector.

Definition at line 162 of file LinearObservationManager.py.

48.24.3.7 def LinearObservationManager::LinearObservationManager::GetName ( self )

Returns the name of the class.

Returns

The name of the class.

Definition at line 224 of file LinearObservationManager.py.

48.24.3.8 def LinearObservationManager::LinearObservationManager::GetNobservation ( self )

Returns the number of available observations.

Returns

The total number of observation at current time.

Definition at line 109 of file LinearObservationManager.py.
def LinearObservationManager::GetObservation (self)

Returns the observations.
This method is called after 'SetTime' set the time at which the observations are requested.

Returns

observation observation vector.

Definition at line 117 of file LinearObservationManager.py.

def LinearObservationManager::GetTangentLinearOperatorRow (self, row)

Returns a row of the tangent linear operator.
This method is called after 'SetTime' set the time at which the operator is defined.

Parameters

in row row index.

Returns

The row row of the tangent linear operator.

Definition at line 188 of file LinearObservationManager.py.

def LinearObservationManager::HasObservation (self, time=None)

Indicates if some observations are available at a given time.

Parameters

in time a given time.

Definition at line 99 of file LinearObservationManager.py.

def LinearObservationManager::Message (self, message)

Receives and handles a message.

Parameters

in message the received message

Definition at line 230 of file LinearObservationManager.py.

def LinearObservationManager::SetTime (self, time)

Sets the current time.

Parameters

in time the current time.

Definition at line 91 of file LinearObservationManager.py.
The documentation for this class was generated from the following file:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/observation_manager/LinearObservationManager.

48.25 Verdandi::Logger Class Reference

Logging class.

#include <Logger.hxx>

Static Public Member Functions

- static void Initialize ()
  Initializes the Logger class.
- static void Initialize (string configuration_file, string section_name)
  Initializes the Logger class.
- static void Initialize ()
  Clears the Logger class.
- static void InitializeOptions ()
  Sets 'options_' to 'default_options_'.
- static void SetFileName (string file_name)
  Sets the name of the log file.
- static void SetOption (int option, bool value)
  Activates or deactivates a specific option.
- static void SetStdout (bool value)
  Activates or deactivates "stdout" option.
- static void SetFile (bool value)
  Activates or deactivates "file" option.
- static void SetUppercase (bool value)
  Activates or deactivates "uppercase" option.
- static void SetLoggingLevel (int level)
  Sets the level of verbosity.
- static void Activate ()
  Activates the Logger.
- static void Deactivate ()
  Deactivates the Logger.
- template<int LEVEL, class T , class S >
  static void Log (const T &object, const S &message, int options=options_)
- template<class T , class S >
  static void Log (const T &object, const S &message, int options=options_)
- template<class T >
  static void Log (const T &object, string message, int options=options_)
- template<class T , class S >
  static void StdOut (const T &object, const S &message)
- template<class T >
  static void StdOut (const T &object, string message)
- static void Command (string command, string parameter, int options=options_)
- static void StdOutCommand (string command, string parameter)
  Reads a specific command and applies the corresponding treatment.
Static Public Attributes

- static const int stdout_ = 1
  Stdout flag.
- static const int file_ = 2
  File flag.
- static const int uppercase_ = 4
  Uppercase flag.

48.25.1 Detailed Description

Logging class.

48.25.2 Member Function Documentation

48.25.2.1 void Verdandi::Logger::Command ( string command, string parameter, int options = options_) [static]

Reads a specific command and applies the corresponding treatment.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>command</td>
<td>string</td>
<td>the specific command.</td>
</tr>
<tr>
<td>parameter</td>
<td>string</td>
<td>the parameter of the command.</td>
</tr>
<tr>
<td>options</td>
<td>int</td>
<td>options.</td>
</tr>
</tbody>
</table>

Note

Only 'hline' command is yet supported.

Definition at line 202 of file Logger.cxx.

48.25.2.2 void Verdandi::Logger::Initialize ( string configuration_file, string section_name ) [static]

Initializes the Logger class.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration_file</td>
<td>string</td>
<td>the configuration file.</td>
</tr>
<tr>
<td>section_name</td>
<td>string</td>
<td>the section in configuration_file where the configuration is to be read.</td>
</tr>
</tbody>
</table>

Definition at line 72 of file Logger.cxx.

48.25.2.3 void Verdandi::Logger::SetFile ( bool value ) [static]

Activates or deactivates "file" option.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>bool</td>
<td>value.</td>
</tr>
</tbody>
</table>

Definition at line 149 of file Logger.cxx.
48.25.2.4 void Verdandi::Logger::SetFileName (string file_name) [static]

Sets the name of the log file.

Parameters

| in | file_name | path to the log file. |

Definition at line 105 of file Logger.cxx.

48.25.2.5 void Verdandi::Logger::SetLoggingLevel (int level) [static]

Sets the level of verbosity.

Parameters

| in | level | level of verbosity. |

Definition at line 173 of file Logger.cxx.

48.25.2.6 void Verdandi::Logger::SetOption (int option, bool value) [static]

Activates or deactivates a specific option.

Parameters

| in | option | option to (des)activate. |
| in | value  | value. |

Definition at line 122 of file Logger.cxx.

48.25.2.7 void Verdandi::Logger::SetStdout (bool value) [static]

Activates or deactivates "stdout" option.

Parameters

| in | value | value. |

Definition at line 137 of file Logger.cxx.

48.25.2.8 void Verdandi::Logger::SetUppercase (bool value) [static]

Activates or deactivates "uppercase" option.

Parameters

| in | value | value. |

Definition at line 161 of file Logger.cxx.

48.25.2.9 void Verdandi::Logger::StdOutCommand (string command, string parameter) [static]

Reads a specific command and applies the corresponding treatment.

The message is always sent to the standard output, and it is possibly written in a log file if the logging level is lower
than or equal to ‘VERDANDI_STDOUT_LOGGING_LEVEL’.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>command</td>
<td>the specific command.</td>
</tr>
<tr>
<td>in</td>
<td>parameter</td>
<td>the parameter of the command.</td>
</tr>
</tbody>
</table>

Note

Only ‘hline’ command is yet supported.

Definition at line 227 of file Logger.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/share/Logger.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/share/Logger.cxx

### 48.26 Verdandi::Lorenz Class Reference

This class is a Lorenz model.

```cpp
#include <Lorenz.hxx>
```

Inheritance diagram for Verdandi::Lorenz:

```
Verdandi::Lorenz
Verdandi::LorenzBase
```

**Public Types**

- typedef T value_type
- typedef T * pointer
- typedef const T * const_pointer
- typedef T & reference
- typedef const T & const_reference
- typedef Matrix< T > state_error_variance
- typedef Vector< T > state_error_variance_row
- typedef Vector< T > state
- typedef Matrix< T > matrix_state_observation

**Public Member Functions**

- **Lorenz ()**
  
  Constructor.
- **Lorenz (string configuration_file)**
  
  Constructor.
- **~Lorenz ()**
  
  Destructor.
- **void Initialize (string configuration_file)**
  
  Initializes the model.
Verdandi::Lorenz Class Reference

- void InitializeStep ()
  
  Initializes the current time step for the model.

- void Forward ()
  
  Advances one step forward in time.

- bool HasFinished () const
  
  Checks whether the model has finished.

- void Save ()
  
  Saves the simulated data.

- void FinalizeStep ()
  
  Finalizes the current time step for the model.

- void Finalize ()
  
  Finalizes the model.

- T GetX () const
  
  Returns the value of X.

- T GetY () const
  
  Returns the value of Y.

- T GetZ () const
  
  Returns the value of Z.

- T GetDelta_t () const
  
  Returns the time step.

- double GetTime () const
  
  Returns the current time.

- void SetTime (double time)
  
  Sets the current time.

- int GetNstate () const
  
  Returns the dimension of the state.

- state & GetState ()
  
  Provides the controlled state vector.

- void StateUpdated ()
  
  Performs some calculations when the update of the model state is done.

- state & GetFullState ()
  
  Provides the full state vector.

- void FullStateUpdated ()
  
  Performs some calculations when the update of the full model state is done.

- state_error_variance_row & GetStateErrorVarianceRow (int row)
  
  Computes a row of the background error covariance matrix B.

- const state_error_variance & GetStateErrorVariance () const
  
  Returns the background error covariance matrix (B) if available.

- bool IsErrorSparse () const
  
  Checks if the error covariance matrix is sparse.

- string GetName () const
  
  Returns the name of the class.

- void Message (string message)
  
  Receives and handles a message.

Static Public Member Functions

- static void StaticMessage (void *object, string message)
  
  Receives and handles a message with a static method.
Protected Attributes

- `T Pr_`
  Prandtl number.
- `T Ra_`
  Rayleigh number.
- `T b_`
  Third parameter.
- `T X_`
  First variable of the system.
- `T Y_`
  Second variable of the system.
- `T Z_`
  Third variable of the system.
- `T X_tmp_`
  Backup of the first variable of the system.
- `T Y_tmp_`
  Backup of the second variable of the system.
- `double Delta_t_`
  Time step.
- `double final_time_`
  Final time of the simulation.
- `double time_`
  Current time.
- `OutputSaver output_saver_`
  Output saver.
- `state state_`
  State vector.

48.26.1 Detailed Description

This class is a Lorenz model.

Template Parameters

| T | the type of floating-point numbers. |

48.26.2 Constructor & Destructor Documentation

48.26.2.1 Verdandi::Lorenz::Lorenz ( string configuration_file )

Constructor.
It reads the initial condition and the time settings.

Parameters

| configuration_file | path to the configuration file. |

Definition at line 50 of file Lorenz.cxx.

48.26.3 Member Function Documentation
48.26.3.1  T Verdandi::Lorenz::GetDelta_t ( ) const

Returns the time step.

Returns
   The time step.

Definition at line 216 of file Lorenz.cxx.

48.26.3.2  Lorenz< T >::state & Verdandi::Lorenz::GetFullState ( )

Provides the full state vector.

Returns
   A reference to the full state vector.

Definition at line 284 of file Lorenz.cxx.

48.26.3.3  string Verdandi::Lorenz::GetName ( ) const [virtual]

Returns the name of the class.

Returns
   The name of the class.

Reimplemented from Verdandi::VerdandiBase.

Definition at line 342 of file Lorenz.cxx.

48.26.3.4  int Verdandi::Lorenz::GetNstate ( ) const

Returns the dimension of the state.

Returns
   The dimension of the state, that is, 3.

Definition at line 249 of file Lorenz.cxx.

48.26.3.5  Lorenz< T >::state & Verdandi::Lorenz::GetState ( )

Provides the controlled state vector.

Returns
   A reference to the state vector.

Definition at line 260 of file Lorenz.cxx.

48.26.3.6  const Lorenz< T >::state_error_variance & Verdandi::Lorenz::GetStateErrorVariance ( ) const

Returns the background error covariance matrix (B) if available.

Returns the background error covariance matrix (B) if available, raises an exception otherwise.
Returns

The matrix of the background error covariance.

Definition at line 320 of file Lorenz.cxx.

48.26.3.7  

Lorenz< T > :: state_error_variance_row & Verdandi::Lorenz::GetStateErrorVarianceRow ( int row )

Computes a row of the background error covariance matrix B.

Parameters

| in | row | row index. |

Returns

The value of row number row.

Definition at line 307 of file Lorenz.cxx.

48.26.3.8  

double Verdandi::Lorenz::GetTime ( ) const

Returns the current time.

Returns

The current time.

Definition at line 227 of file Lorenz.cxx.

48.26.3.9  

T Verdandi::Lorenz::GetX ( ) const

Returns the value of X.

Returns

The value of X.

Definition at line 183 of file Lorenz.cxx.

48.26.3.10  

T Verdandi::Lorenz::GetY ( ) const

Returns the value of Y.

Returns

The value of Y.

Definition at line 194 of file Lorenz.cxx.

48.26.3.11  

T Verdandi::Lorenz::GetZ ( ) const

Returns the value of Z.

Returns

The value of Z.

Definition at line 205 of file Lorenz.cxx.
48.26.3.12 bool Verdandi::Lorenz::HasFinished ( ) const

Checks whether the model has finished.

Returns

True if no more data assimilation is required, false otherwise.

Definition at line 141 of file Lorenz.cxx.

48.26.3.13 void Verdandi::Lorenz::Initialize ( string configuration_file )

Initializes the model.

It reads the initial condition and the time settings.

Parameters

| in | configuration_file | configuration file. |

Definition at line 73 of file Lorenz.cxx.

48.26.3.14 bool Verdandi::Lorenz::IsErrorSparse ( ) const

Checks if the error covariance matrix is sparse.

Returns

True if there is a sparse error matrix, false otherwise.

Definition at line 331 of file Lorenz.cxx.

48.26.3.15 void Verdandi::Lorenz::Save ( )

Saves the simulated data.

It saves the state.

Definition at line 151 of file Lorenz.cxx.

48.26.3.16 void Verdandi::Lorenz::SetTime ( double time )

Sets the current time.

Parameters

| in | time | the current time. |

Definition at line 238 of file Lorenz.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/model/Lorenz.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/model/Lorenz.cxx
This class enables objects to communicate using messages.

```cpp
#include <MessageHandler.hxx>
```

### Public Types

- `typedef void (*)(function_pointer)(void *, string)`
  - Function pointer.
- `typedef list<pair<void *, function_pointer>> recipient_list`
  - Recipient list.
- `typedef map<string, recipient_list> recipient_map`
  - Recipient map.

### Static Public Member Functions

- `template<class R> static void AddRecipient(string recipient, R &object, function_pointer pointer) [static]`
  - Adds a new object in the recipient list.
- `static void AddRecipient(string recipient, void *object, function_pointer pointer)`
- `static void Send(string recipient, string message)`
  - Sends a message to a recipient.
- `template<class Sender> static void Send(const Sender &sender, string recipient, string message)`
- `static string GetName()`
  - Returns the name of the class, that is, "MessageHandler".
- `template<class R> static void RemoveRecipient(R &object)`

### Detailed Description

This class enables objects to communicate using messages.

### Member Function Documentation

#### 48.27.2.1 void Verdandi::MessageHandler::AddRecipient ( string recipient, void *object, 
MessageHandler::function_pointer pointer ) [static]

Adds a new object in the recipient list.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>recipient</code></td>
<td>the string describing the object to add.</td>
</tr>
<tr>
<td><code>object</code></td>
<td>pointer to the recipient object.</td>
</tr>
<tr>
<td><code>pointer</code></td>
<td>the pointer to the method to add.</td>
</tr>
</tbody>
</table>

Definition at line 43 of file MessageHandler.cxx.

#### 48.27.2.2 string Verdandi::MessageHandler::GetName ( ) [static]

Returns the name of the class, that is, "MessageHandler".
48.28 Verdandi::ModelTemplate Class Reference

Returns

The name of the class.

Definition at line 79 of file MessageHandler.cxx.

48.27.2.3 void Verdandi::MessageHandler::Send ( string recipient, string message ) [static]

Sends a message to a recipient.

Parameters

| in | recipient | the recipient of the message. |
| in | message   | the string containing the message. |

Definition at line 58 of file MessageHandler.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/share/MessageHandler.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/share/MessageHandler.cxx

48.28 Verdandi::ModelTemplate Class Reference

This class is a model template.

#include <ModelTemplate.hxx>

Inheritance diagram for Verdandi::ModelTemplate:

Verdandi::ModelTemplate

Verdandi::VerdandiBase

Public Types

- typedef double value_type
  
  The numerical type (e.g., double).
- typedef Matrix< double > state_error_variance
  
  Type of the state error variance.
- typedef Matrix< double > state_error_variance_reduced
  
  Type of the reduced matrix $U$ in the $LU^T$ decomposition of the state error variance.
- typedef Vector< double > state_error_variance_row
  
  Type of a row of the state error variance.
- typedef Matrix< double > error_variance
  
  Type of the model error variance.
- typedef Matrix< double > matrix_state_observation
  
  Type of the state/observation crossed matrix.
- typedef Matrix< double > tangent_linear_operator
  
  Type of the tangent linear model.
- typedef Vector< double > state
  
  Type of the state vector.
typedef Vector< double > uncertain_parameter
Type of an uncertain parameter.

typedef Matrix< double, Symmetric, RowSymPacked > parameter_variance
Type of the uncertain parameter covariance matrix.

Public Member Functions

- ModelTemplate ()
  Constructor.
- ~ModelTemplate ()
  Destructor.
- void Initialize (string configuration_file)
  Initializes the model.
- void InitializeStep ()
  Initializes the current time step for the model.
- void Forward ()
  Advances one step forward in time.
- void BackwardAdjoint (state &observation_term)
  Performs one step backward in adjoint model.
- bool HasFinished () const
  Checks whether the model has finished.
- void FinalizeStep ()
  Finalizes the current time step for the model.
- void Finalize ()
  Finalizes the model.
- double ApplyOperator (state &x, bool preserve_state=true)
  Applies the model to a given vector.
- double ApplyTangentLinearOperator (state &x)
  Applies the tangent linear model to a given vector.
- tangent_linear_operator& GetTangentLinearOperator ()
  Gets the tangent linear model.
- double GetTime ()
  Returns the current time.
- void SetTime (double time)
  Sets the time of the model to a given time.
- int GetNstate ()
  Returns the state vector size.
- int GetNfull_state ()
  Returns the size of the full state vector.
- state & GetState ()
  Provides the controlled state vector.
- void StateUpdated ()
  Carries out some calculations when the model state has been updated.
- state & GetFullState ()
  Provides the full state vector.
- void FullStateUpdated ()
  Carries out some calculations when the model state has been updated.
- state & GetStateLowerBound ()
  Provides the state lower bound.
- state & GetStateUpperBound ()
Provides the state upper bound.

- `state & GetAdjointState ()`
  Returns the adjoint state vector.
- `void AdjointStateUpdated ()`
- `state & GetAdditionalAdjointTerm ()`
  Returns a term that will be added to the adjoint state.
- `double GetAdditionalCostTerm ()`
  Returns a term that will be added to the cost function.
- `int GetNparameter ()`
  Returns the number of parameters to be perturbed.
- `uncertain_parameter & GetParameter (int i)`
  Gets the i-th uncertain parameter.
- `void ParameterUpdated (int i)`
  Carries out some calculations when the i-th parameter has been updated.
- `string GetParameterName (int i)`
  Returns the name of a parameter to be perturbed.
- `Vector< double > & GetParameterCorrelation (int i)`
  Returns the correlation between the uncertain parameters.
- `string GetParameterPDF (int i)`
  Returns the PDF of the i-th parameter.
- `parameter_variance & GetParameterVariance (int i)`
  Returns the covariance matrix associated with the i-th parameter.
- `Vector< double > & GetParameterPDFData (int i)`
  Returns parameters associated with the PDF of some model parameter.
- `string GetParameterOption (int i)`
  Returns the perturbation option of the i-th parameter.
- `state_error_variance_row & GetStateErrorVarianceRow (int row)`
  Computes a row of the variance of the state error.
- `state_error_variance & GetStateErrorVariance ()`
  Returns the state error variance.
- `error_variance & GetErrorVariance ()`
  Returns the model error variance.
- `error_variance & GetErrorVarianceSqrt ()`
  Returns the square root of the model error variance.
- `state_error_variance & GetStateErrorVarianceProjector ()`
- `state_error_variance_reduced & GetStateErrorVarianceReduced ()`
- `const state_error_variance & GetStateErrorVarianceInverse () const`
  Returns the inverse of the background error variance ($B^{-1}$).
- `string GetName () const`
  Returns the name of the class.
- `void Message (string message)`
  Receives and handles a message.

**Static Public Member Functions**

- static void StaticMessage (void *object, string message)
  Receives and handles a message with a static method.

**48.28.1 Detailed Description**

This class is a model template.
48.28.2 Member Function Documentation

48.28.2.1 void Verdandi::ModelTemplate::AdjointStateUpdated ( )

Carries out some calculations when the adjoint state has been updated.
Definition at line 303 of file ModelTemplate.cxx.

48.28.2.2 double Verdandi::ModelTemplate::ApplyOperator ( state & x, bool preserve_state = true )

Applies the model to a given vector.
The current state of the model is modified.

Parameters

<table>
<thead>
<tr>
<th>in, out</th>
<th>x</th>
<th>on entry, the state vector to which the model should be applied; on exit, the result.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>preserve_state</td>
<td>Boolean to indicate if the model state has to be preserved.</td>
</tr>
</tbody>
</table>

Returns

The time associated with x on exit plus one time step.

Warning

The time of the model has to be preserved.
Definition at line 151 of file ModelTemplate.cxx.

48.28.2.3 double Verdandi::ModelTemplate::ApplyTangentLinearOperator ( state & x )

Applies the tangent linear model to a given vector.

Parameters

<table>
<thead>
<tr>
<th>in, out</th>
<th>x</th>
<th>on entry, the state vector to which the tangent linear model should be applied; on exit, the result.</th>
</tr>
</thead>
</table>

Returns

The time associated with x on exit. This time should be the model time plus one time step.
Definition at line 165 of file ModelTemplate.cxx.

48.28.2.4 void Verdandi::ModelTemplate::BackwardAdjoint ( state & observation_term )

Performs one step backward in adjoint model.

Parameters

| in       | observation_term | \( H^T R^{-1} (y - Hx) \). |

Definition at line 90 of file ModelTemplate.cxx.
48.28.2.5  void Verdandi::ModelTemplate::Forward ( )

Advances one step forward in time.

\[ x_{h+1}^f = \mathcal{M}_h(x_h, p_h) \, . \]

Definition at line 80 of file ModelTemplate.cxx.

48.28.2.6  ModelTemplate::state & Verdandi::ModelTemplate::GetAdditionalAdjointTerm ( )

Returns a term that will be added to the adjoint state.

Returns
   The additional adjoint term.

Definition at line 314 of file ModelTemplate.cxx.

48.28.2.7  double Verdandi::ModelTemplate::GetAdditionalCostTerm ( )

Returns a term that will be added to the cost function.

Returns
   The additional cost term.

Definition at line 324 of file ModelTemplate.cxx.

48.28.2.8  ModelTemplate::state & Verdandi::ModelTemplate::GetAdjointState ( )

Returns the adjoint state vector.

Returns
   The adjoint state vector.

Definition at line 293 of file ModelTemplate.cxx.

48.28.2.9  ModelTemplate::error_variance & Verdandi::ModelTemplate::GetErrorVariance ( )

Returns the model error variance.

Returns
   The model error variance.

Note
   If the matrix is empty, it is then assumed there is no model error.

Definition at line 520 of file ModelTemplate.cxx.
Returns the square root of the model error variance.

Definition at line 508 of file ModelTemplate.cxx.

Provides the full state vector.

state the controlled state vector.

Definition at line 274 of file ModelTemplate.cxx.

Returns the name of the class.

The name of the class.

Reimplemented from Verdandi::VerdandiBase.

Definition at line 530 of file ModelTemplate.cxx.

Returns the size of the full state vector.

The size of the full state vector.

Definition at line 223 of file ModelTemplate.cxx.

Returns the number of parameters to be perturbed.

The number of parameters to be perturbed.

Definition at line 334 of file ModelTemplate.cxx.

Returns the state vector size.

The state vector size.

Definition at line 213 of file ModelTemplate.cxx.
48.28.2.16  ModelTemplate::uncertain_parameter & Verdandi::ModelTemplate::GetParameter ( int i )

Gets the i-th uncertain parameter.

Parameters

| in | i | index of the parameter. |

Returns

The vector associated with the i-th parameter.

Definition at line 345 of file ModelTemplate.cxx.

48.28.2.17  Vector< double > & Verdandi::ModelTemplate::GetParameterCorrelation ( int i )

Returns the correlation between the uncertain parameters.

Parameters

| in | i | parameter index. |

Returns

The correlation between the uncertain parameters.

Definition at line 379 of file ModelTemplate.cxx.

48.28.2.18  string Verdandi::ModelTemplate::GetParameterName ( int i )

Returns the name of a parameter to be perturbed.

Parameters

| in | i | index of the parameter. |

Returns

The name of the parameter.

Definition at line 368 of file ModelTemplate.cxx.

48.28.2.19  string Verdandi::ModelTemplate::GetParameterOption ( int i )

Returns the perturbation option of the i-th parameter.

Parameters

| in | i | parameter index. |

Returns

The perturbation option of the i-th parameter.

Definition at line 425 of file ModelTemplate.cxx.
string Verdandi::ModelTemplate::GetParameterPDF ( int i )

Returns the PDF of the i-th parameter.

Parameters

| in | i | uncertain-variable index. |

Returns

The PDF of the i-th parameter.

Definition at line 390 of file ModelTemplate.cxx.

Vector< double > & Verdandi::ModelTemplate::GetParameterPDFData ( int i )

Returns parameters associated with the PDF of some model parameter.
In case of normal or log-normal distribution, the parameters are clipping parameters.

Parameters

| in | i | model parameter index. |

Returns

The parameters associated with the i-th parameter.

Definition at line 414 of file ModelTemplate.cxx.

ModelTemplate::parameter_variance & Verdandi::ModelTemplate::GetParameterVariance ( int i )

Returns the covariance matrix associated with the i-th parameter.

Parameters

| in | i | parameter index. |

Returns

The covariance matrix associated with the i-th parameter.

Definition at line 402 of file ModelTemplate.cxx.

ModelTemplate::state & Verdandi::ModelTemplate::GetState ( )

Provides the controlled state vector.

Returns

state the controlled state vector.

Definition at line 233 of file ModelTemplate.cxx.

ModelTemplate::state_error_variance & Verdandi::ModelTemplate::GetStateErrorVariance ( )

Returns the state error variance.
Returns

The state error variance.

Definition at line 455 of file ModelTemplate.cxx.

48.28.2.25  const ModelTemplate::state_error_variance & Verdandi::ModelTemplate::GetStateErrorVarianceInverse ( )

Returns the inverse of the background error variance (\(B^{-1}\)).

Returns

The inverse of the background error variance (\(B^{-1}\)).

Definition at line 497 of file ModelTemplate.cxx.

48.28.2.26  ModelTemplate::state_error_variance & Verdandi::ModelTemplate::GetStateErrorVarianceProjector ( )

Returns the matrix L in the decomposition of the state error covariance matrix (\(B\)) as a product \(LUU^T\).

Returns

The matrix \(L\).

Definition at line 469 of file ModelTemplate.cxx.

48.28.2.27  ModelTemplate::state_error_variance_reduced & Verdandi::ModelTemplate::GetStateErrorVarianceReduced ( )

Returns the matrix U in the decomposition of the state error covariance matrix (\(B\)) as a product \(LUU^T\).

Returns

The matrix \(U\).

Definition at line 484 of file ModelTemplate.cxx.

48.28.2.28  ModelTemplate::state_error_variance_row & Verdandi::ModelTemplate::GetStateErrorVarianceRow ( int row )

Computes a row of the variance of the state error.

Parameters

| in | row | row index. |

Returns

The row with index \(row\) in the state error variance.

Definition at line 442 of file ModelTemplate.cxx.

48.28.2.29  ModelTemplate::state & Verdandi::ModelTemplate::GetStateLowerBound ( )

Provides the state lower bound.
Returns

The state lower bound (componentwise).

Definition at line 252 of file ModelTemplate.cxx.

48.28.2.30 ModelTemplate::state & Verdandi::ModelTemplate::GetStateUpperBound ( )

Provides the state upper bound.

Returns

The state upper bound (componentwise).

Definition at line 263 of file ModelTemplate.cxx.

48.28.2.31 ModelTemplate::tangent_linear_operator & Verdandi::ModelTemplate::GetTangentLinearOperator ( )

Gets the tangent linear model.

Parameters

| out | A | the matrix of the tangent linear model. |

Definition at line 177 of file ModelTemplate.cxx.

48.28.2.32 double Verdandi::ModelTemplate::GetTime ( )

Returns the current time.

Returns

The current time.

Definition at line 193 of file ModelTemplate.cxx.

48.28.2.33 bool Verdandi::ModelTemplate::HasFinished ( ) const

Checks whether the model has finished.

Returns

True if the simulation is done, false otherwise.

Definition at line 100 of file ModelTemplate.cxx.

48.28.2.34 void Verdandi::ModelTemplate::Initialize ( string configuration_file )

Initializes the model.

Parameters

| in | configuration_file | configuration file. |

Definition at line 59 of file ModelTemplate.cxx.
48.28.2.35 void Verdandi::ModelTemplate::ParameterUpdated ( int i )

Carries out some calculations when the i-th parameter has been updated.

Parameters

| in  | i  | index of the parameter. |

Definition at line 357 of file ModelTemplate.cxx.

48.28.2.36 void Verdandi::ModelTemplate::SetTime ( double time )

Sets the time of the model to a given time.

Parameters

| in  | time | a given time. |

Definition at line 203 of file ModelTemplate.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/model/ModelTemplate.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/model/ModelTemplate.cxx

48.29 Verdandi::MonteCarlo Class Reference

This class performs allows to perform Monte Carlo simulations.

#include <MonteCarlo.hxx>

Inheritance diagram for Verdandi::MonteCarlo:

```
Verdandi::VerdandiBase
  /
Verdandi::MonteCarlo
```

Public Types

- typedef Model::state model_state
  Type of the model state vector.
- typedef Model::state::value_type Ts
  Value type of the model state vector.
- typedef Model::uncertain_parameter uncertain_parameter
  Type of the uncertain parameter.
- typedef Model::uncertain_parameter::value_type Tp
  Value type of the uncertain parameter.

Public Member Functions

- MonteCarlo ()
  Main constructor.
- `~MonteCarlo()`
  Destructor.
- `template<class T0, class Storage0, class Allocator0>
  void Clear(Vector<T0, Storage0, Allocator0> &V)`
  Clears a vector.
- `template<class T0, class Allocator0>
  void Clear(Vector<T0, Collection, Allocator0> &V)`
  Clears a vector collection.
- `template<class T0, class Storage0, class Allocator0>
  void SetDimension(Vector<T0, Storage0, Allocator0> &in, Vector<T0, Storage0, Allocator0> &out)`
  Allocates an output vector to the dimension of the input vector.
- `template<class T0, class Allocator0>
  void SetDimension(Vector<T0, Collection, Allocator0> &in, Vector<T0, Collection, Allocator0> &out)`
  Allocates an output vector collection to the dimension of the input vector collection.
- `template<class T0, class Allocator0>
  void Fill(Vector<T0, Collection, Allocator0> &in, string pdf)`
  Fills an input vector collection according to its probability distribution.
- `template<class T0, class Storage0, class Allocator0>
  void Fill(Vector<T0, Storage0, Allocator0> &in, string pdf)`
  Fills an input vector according to its probability distribution.
- `void Initialize(string configuration_file, bool initialize_model=true, bool initialize_perturbation_manager=true)`
  Initializes the simulation.
- `void Initialize(VerdandiOps &configuration, bool initialize_model=true, bool initialize_perturbation_manager=true)`
  Initializes the simulation.
- `void InitializeStep()`
  Initializes the model before a time step.
- `void Forward()`
  Performs a step forward without optimal interpolation.
- `void FinalizeStep()`
  Finalizes a step for the model.
- `void Finalize()`
  Finalizes the model.
- `bool HasFinished()`
  Checks whether the model has finished.
- `Model & GetModel()`
  Returns the model.
- `OutputSaver & GetOutputSaver()`
  Returns the output saver.
- `string GetName() const`
  Returns the name of the class.
- `void Message(string message)`
  Receives and handles a message.

Static Public Member Functions

- static void StaticMessage(void *object, string message)
  Receives and handles a message with a static method.
Protected Attributes

- Model `model_`
  Underlying model.
- PerturbationManager `perturbation_manager_`
  Perturbation managers.
- `vector< uncertain_parameter > perturbation_`
  Perturbations vectors.
- `int iteration_`
  Iteration.
- `Vector< double > time_`
  Time vector.
- `string configuration_file_`
  Path to the configuration file.
- `string model_configuration_file_`
  Path to the model configuration file.
- `string perturbation_manager_configuration_file_`
  Path to the configuration file for the perturbation manager.
- `map< string, bool > option_display_`
  Display options.
- `string perturbation_source_`
  Source of the perturbations: "file" or "random".
- `string perturbation_file_`
  In case `perturbation_source_` is "file", path to file storing the perturbations. It should contain 
  "&p" to be replaced with the parameter name.
- `OutputSaver output_saver_`
  Output saver.

48.29.1 Detailed Description

This class performs allows to perform Monte Carlo simulations.

The class performs a single simulation with perturbed data. In order to complete a full Monte Carlo simulation, one has to launch several simulations using this class.

48.29.2 Constructor & Destructor Documentation

48.29.2.1 Verdandi::MonteCarlo::MonteCarlo ( )

Main constructor.
Builds the driver and reads option keys in the configuration file.

Parameters

| in | configuration_file | configuration file. |

Definition at line 40 of file MonteCarlo.cxx.

48.29.3 Member Function Documentation
48.29.3.1  template<class T0 , class Storage0 , class Allocator0 > void Verdandi::MonteCarlo::Clear ( Vector<T0, Storage0, Allocator0> & V )

Clears a vector.

Parameters

| in, out | V | vector to be cleared. |

Definition at line 67 of file MonteCarlo.cxx.

48.29.3.2  template<class T0 , class Allocator0 > void Verdandi::MonteCarlo::Clear ( Vector<T0, Collection, Allocator0> & V )

Clears a vector collection.

Each inner vector of the collection is cleared.

Parameters

| in, out | V | vector to be cleared. |

Definition at line 80 of file MonteCarlo.cxx.

48.29.3.3  template<class T0 , class Allocator0 > void Verdandi::MonteCarlo::Fill ( Vector<T0, Collection, Allocator0> & in, string pdf )

Fills an input vector collection according to its probability distribution.

Parameters

| in, out | in | input collection vector. |
| in | pdf | probability density function: Normal, NormalHomogeneous, LogNormal or LogNormalHomogeneous. |

Definition at line 139 of file MonteCarlo.cxx.

48.29.3.4  template<class T0 , class Storage0 , class Allocator0 > void Verdandi::MonteCarlo::Fill ( Vector<T0, Storage0, Allocator0> & in, string pdf )

Fills an input vector according to its probability distribution.

Parameters

| in, out | in | input vector. |
| in | pdf | probability density function: Normal, NormalHomogeneous, LogNormal or LogNormalHomogeneous. |

Definition at line 161 of file MonteCarlo.cxx.

48.29.3.5  Model & Verdandi::MonteCarlo::GetModel ( )

Returns the model.
Returns

The model.

Definition at line 494 of file MonteCarlo.cxx.

48.29.3.6 string Verdandi::MonteCarlo::GetName() const [virtual]

Returns the name of the class.

Returns

The name of the class.

Reimplemented from Verdandi::VerdandiBase.

Definition at line 517 of file MonteCarlo.cxx.

48.29.3.7 OutputSaver & Verdandi::MonteCarlo::GetOutputSaver()

Returns the output saver.

Returns

The output saver.

Definition at line 506 of file MonteCarlo.cxx.

48.29.3.8 bool Verdandi::MonteCarlo::HasFinished()

Checks whether the model has finished.

Returns

True if the simulation is finished, false otherwise.

Definition at line 477 of file MonteCarlo.cxx.

48.29.3.9 void Verdandi::MonteCarlo::Initialize(string configuration_file, bool initialize_model = true, bool initialize_perturbation_manager = true)

Initializes the simulation.

It reads the configuration and initializes the model and the perturbation manager.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>configuration_file</th>
<th>configuration file for the method.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>initialize_model</td>
<td>should the model be initialized with a call to Model::Initialize(string)?</td>
</tr>
<tr>
<td>in</td>
<td>initialize_perturbation_manager</td>
<td>should the perturbation manager be initialized with a call to PerturbationManager::Initialize(string)?</td>
</tr>
</tbody>
</table>

Warning

If initialize_model is set to false, the model should be initialized before calling this function.

Definition at line 184 of file MonteCarlo.cxx.
void Verdandi::MonteCarlo::Initialize ( VerandDiOps & configuration, bool initialize_model = true, bool initialize_perturbation_manager = true )

Initializes the simulation.

It reads the configuration and initializes the model and the perturbation manager.

### Parameters

<table>
<thead>
<tr>
<th></th>
<th>configuration_file</th>
<th>configuration file for the method.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>initialize_model</td>
<td>should the model be initialized with a call to Model::Initialize(string)?</td>
</tr>
<tr>
<td></td>
<td>initialize_perturbation_manager</td>
<td>should the perturbation manager be initialized with a call to PerturbationManager::Initialize(string)?</td>
</tr>
</tbody>
</table>

**Warning**

If `initialize_model` is set to false, the model should be initialized before calling this function.

Definition at line 208 of file MonteCarlo.cxx.

```cpp
template<class T0 , class Storage0 , class Allocator0 >
void Verdandi::MonteCarlo::SetDimension ( Vector<T0, Storage0, Allocator0> & in, Vector<T0, Storage0, Allocator0> & out )
```

Allocates an output vector to the dimension of the input vector.

### Parameters

<table>
<thead>
<tr>
<th></th>
<th>in</th>
<th>input vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>out</td>
<td>output vector.</td>
</tr>
</tbody>
</table>

Definition at line 100 of file MonteCarlo.cxx.

```cpp
template<class T0 , class Allocator0 >
void Verdandi::MonteCarlo::SetDimension ( Vector<T0, Collection, Allocator0> & in, Vector<T0, Collection, Allocator0> & out )
```

Allocates an output vector collection to the dimension of the input vector collection.

### Parameters

<table>
<thead>
<tr>
<th></th>
<th>in</th>
<th>input collection vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>out</td>
<td>output collection vector.</td>
</tr>
</tbody>
</table>

Definition at line 116 of file MonteCarlo.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/MonteCarlo.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/MonteCarlo.cxx
Public Member Functions

- `Verdandi::NewranPerturbationManager()`
  Default constructor.

- `Verdandi::NewranPerturbationManager(string configuration_file)`
  Main constructor.

- `~Verdandi::NewranPerturbationManager()`
  Destructor.

- `void Initialize(string configuration_file)`
  Initializes the manager.

- `void Initialize(VerdandiOps &configuration_stream)`
  Initializes the manager.

- `void Reinitialize()`
  Reinitializes the manager.

- `void Finalize()`
  Copies the seed to disk.

- `double Normal(double mean, double variance, Vector< double, VectFull > &parameter)`
  Generates a random number with a normal distribution.

- `double LogNormal(double mean, double variance, Vector< double, VectFull > &parameter)`
  Generates a random number with a log-normal distribution.

- `template <class T0, class T1, class Prop0, class Allocator0>
  void Normal(Matrix< T0, Prop0, RowSymPacked, Allocator0 > variance, Vector< double, VectFull > &parameter, Vector< T1, VectFull, Allocator0 > &sample)`
  Generates a vector of random numbers with normal distribution.

- `template <class T0, class T1, class Prop0, class Allocator0, class T1, class Allocator1>
  void LogNormal(Matrix< T0, Prop0, RowSymPacked, Allocator0 > variance, Vector< double, VectFull > &parameter, Vector< T1, VectFull, Allocator1 > &output)`
  Generate a random vector with a log-normal distribution.

- `template <class T0, class T1, class Allocator1>
  void NormalHomogeneous(T0 variance, Vector< double, VectFull > &parameter, Vector< T1, VectFull, Allocator1 > &output)`
  Generate a random vector with a homogeneous normal distribution.

- `template <class T0, class T1, class Allocator1>
  void LogNormalHomogeneous(T0 variance, Vector< double, VectFull > &parameter, Vector< T1, VectFull, Allocator1 > &output)`
  Generates a random vector with a homogeneous log normal distribution.

- `template <class T0, class T1, class Allocator0>
  bool NormalClipping(Vector< T0, VectFull > &diagonal, Vector< double, VectFull > &parameter, Vector< T1, VectFull, Allocator0 > &output)`
  Tests if a vector satisfies clipping constraints.

- `template <class T0, class T1, class Allocator1>
  bool NormalClipping(Vector< T0, VectFull > &diagonal, Vector< double, VectFull > &parameter, Vector< T1, VectFull, Allocator1 > &output)`
  Tests if a vector satisfies clipping constraints.
- void Sample (string pdf, Matrix<T0, Prop0, RowSymPacked, Allocator0> &variance, Vector<double, VectFull> &parameter, Vector<double, VectFull> &correlation, Vector<T1, Collection, Allocator1> &output)

- void Sample (string pdf, T0 variance, Vector<double, VectFull> &parameter, Vector<double, VectFull> &correlation, Vector<T1, VectFull, Allocator1> &output)

- void Sample (string pdf, T0 variance, Vector<double, VectFull> &parameter, Vector<double, VectFull> &correlation, Vector<T1, Collection, Allocator1> &output)

Protected Attributes

- NEWRAN::LGM_mixed ∗ urng_
  Uniform random number generator.

- string seed_type_
  String that defines how the seed is initialized: "time", "number" or "directory".

- double seed_number_
  Newran seed number.

- string seed_directory_
  Newran seed directory.

48.30.1 Detailed Description

This class generates random samples using Newran.

48.30.2 Constructor & Destructor Documentation

48.30.2.1 Verdandi::NewranPerturbationManager::NewranPerturbationManager ( )

Default constructor.
The seed is initialized from the system clock.
Definition at line 43 of file NewranPerturbationManager.cxx.

48.30.2.2 Verdandi::NewranPerturbationManager::NewranPerturbationManager ( string configuration_file )

Main constructor.
Builds the manager and reads option keys in the configuration file.
Parameters

| in | configuration_file | configuration file. |

Definition at line 58 of file NewranPerturbationManager.cxx.

48.30.3 Member Function Documentation

48.30.3.1 void Verdandi::NewranPerturbationManager::Finalize ( )

Copies the seed to disk.
Saves and unlocks the seed.
Definition at line 147 of file NewranPerturbationManager.cxx.
48.30.3.2  void Verdandi::NewranPerturbationManager::Initialize ( string configuration_file )

Initializes the manager.

Parameters

| in | configuration_file | configuration file. |

Reimplemented from Verdandi::BasePerturbationManager< NewranPerturbationManager >.

Definition at line 81 of file NewranPerturbationManager.cxx.

48.30.3.3  void Verdandi::NewranPerturbationManager::Initialize ( VerdandiOps & configuration_stream )

Initializes the manager.

Parameters

| in | configuration_stream | configuration stream. |

Definition at line 92 of file NewranPerturbationManager.cxx.

48.30.3.4  double Verdandi::NewranPerturbationManager::LogNormal ( double mean, double variance, Vector< double, VectFull > & parameter )

Generates a random number with a log-normal distribution.

Parameters

| in | mean | mean of the normal distribution. |
| in | variance | variance of the random variable. |
| in | parameter | vector of parameters. The vector may either be empty or contain two clipping parameters \((a,b)\). With the clipping parameters, for a normal distribution, any random value lies in \([\mu - a\sigma, \mu + b\sigma]\) where \(\mu\) is the mean of the random variable and \(\sigma\) is its standard deviation. |

Returns

A random number following the previously described normal distribution.

Definition at line 206 of file NewranPerturbationManager.cxx.

48.30.3.5  double Verdandi::NewranPerturbationManager::Normal ( double mean, double variance, Vector< double, VectFull > & parameter )

Generates a random number with a normal distribution.

Parameters

| in | mean | mean of the normal distribution. |
| in | variance | variance of the random variable. |
| in | parameter | vector of parameters. The vector may either be empty or contain two clipping parameters \((a,b)\). With the clipping parameters, for a normal distribution, any random value lies in \([\mu - a\sigma, \mu + b\sigma]\) where \(\mu\) is the mean of the random variable and \(\sigma\) is its standard deviation. |
Returns

A random number following the previously described normal distribution.

Definition at line 174 of file NewranPerturbationManager.cxx.

48.30.3.6 template<class T0 , class T1 , class Prop0 , class Allocator0 > void Verdandi::NewranPerturbationManager::Normal ( Matrix<T0, Prop0, RowSymPacked, Allocator0 > variance, Vector<double, VectFull > & parameter, Vector<T1, VectFull, Allocator0 > & output )

Generates a vector of random numbers with normal distribution.
Each component of the random vector is generated independently.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>variance</th>
<th>variance of the random variable.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>parameter</td>
<td>vector of parameters. The vector may either be empty or contain two clipping parameters ((a,b)). With the clipping parameters, for a normal distribution, any random value lies in ([\mu - a \sigma, \mu + b \sigma]) where (\mu) is the mean of the random variable and (\sigma) is its standard deviation.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>the generated random vector.</td>
</tr>
</tbody>
</table>

Definition at line 233 of file NewranPerturbationManager.cxx.

48.30.3.7 template<class T0 , class T1 , class Allocator1 > bool Verdandi::NewranPerturbationManager::NormalClipping ( Vector<T0, VectFull > & diagonal, Vector<double, VectFull > & parameter, Vector<T1, VectFull, Allocator1 > & output )

Tests if a vector satisfies clipping constraints.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>diagonal</th>
<th>diagonal coefficients of the covariance matrix.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>permutation</td>
<td>vector of permutations done during the Cholesky decomposition.</td>
</tr>
<tr>
<td>in</td>
<td>parameter</td>
<td>vector of parameters. The vector may either be empty or contain two clipping parameters ((a,b)). With the clipping parameters, for a normal distribution, any random value lies in ([\mu - a \sigma, \mu + b \sigma]) where (\mu) is the mean of the random variable and (\sigma) is its standard deviation.</td>
</tr>
<tr>
<td>in</td>
<td>output</td>
<td>vector to be tested. This vector was generated using a covariance matrix with diagonal diagonal.</td>
</tr>
</tbody>
</table>

Returns

true if the vector satisfies the constraints.

Definition at line 385 of file NewranPerturbationManager.cxx.

48.30.3.8 void Verdandi::NewranPerturbationManager::Reinitialize ( )

Reinitializes the manager.
Locks and reloads the seed.

Definition at line 129 of file NewranPerturbationManager.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/NewranPerturbationManager.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/NewranPerturbationManager.cxx
48.31 Verdandi::ObservationAggregator Class Reference

Observation manager which computes an aggregated observation vector.

#include <ObservationAggregator.hxx>

Public Member Functions

- **ObservationAggregator ()**
  
  Default constructor.

- **ObservationAggregator (string configuration_file)**
  
  Constructor.

- **~ObservationAggregator ()**
  
  Destructor.

- **void Initialize (string configuration_file)**

  Initializer.

- **void DiscardObservation (bool discard_observation)**

  Activates or deactivates the option 'discard_observation'.

- **void GetContributionInterval (double time, double &time_inf, double &time_sup, int &selection_policy) const**

  Returns the contribution time interval corresponding to a given time.

- **template <class time_vector, class observation_vector2, class observation_vector>
  void Aggregate (const time_vector &observation_time, const Vector<double> &contribution, const observation_vector2 &observation, double time, observation_vector &aggregated_observation)**

  Computes an aggregated observation vector over a list of times.

- **template <class time_vector, class variable_vector2, class observation_vector3, class variable_vector, class observation_vector2>
  void Aggregate (const time_vector &observation_time, const Vector<double> &contribution, const variable_vector2 &observation_variable, const observation_vector3 &observation, double time, variable_vector &aggregated_variable, observation_vector2 &aggregated_observation)**

  Computes aggregated observation over a list of times and observations.

- **template <class time_vector, class variable_vector2, class index_vector3, class observation_vector3, class variable_vector, class index_vector2, class observation_vector2>
  void Aggregate (const time_vector &observation_time, const Vector<double> &contribution, const variable_vector2 &observation_variable, const index_vector3 &observation_index, const observation_vector3 &observation, double time, variable_vector &aggregated_variable, index_vector2 &aggregated_index, observation_vector2 &aggregated_observation)**

  Computes aggregated observation over a list of times and observations.

- **int CreateTrack ()**

  Creates a new track.

- **void SetTrack (int track)**

  Sets the active track to a given track.

- **double LastTime () const**

  Returns the last time of the current track.

- **double LastTime (int track) const**

  Returns the last time of a given track.

- **void PushTime (double time)**

  Pushes a time to the current track.

- **void PushTime (double time, int track)**

  Pushes a time to a given track.

- **template <class time_vector>
  void Contribution (double time, const time_vector &observation_time, Vector<double> &contribution)**

  Computes the contributions of given observations at a given time.
template<class time_vector>
void Contribution (double time, const time_vector &observation_time, Vector<double> &width_left, Vector<double> &width_right, Vector<double> &contribution)

Computes the contributions of given observations at a given time.

– double Contribution (double delta_t) const

Computes the contribution of an observation given a time difference.

– template<class time_vector>
void GetValueIndex (time_vector &X, double value, int &index_inf, int &index_sup) const

Returns the least interval containing a given value.

– string GetName () const

Returns the name of the class.

48.31.1 Detailed Description

Observation manager which computes an aggregated observation vector.

Template Parameters

\( T \) the type of floating-point numbers.

48.31.2 Constructor & Destructor Documentation

48.31.2.1 Verdandi::ObservationAggregator::ObservationAggregator ( string configuration_file )

Constructor.

Parameters

\textbf{in} configuration_file configuration file.

Definition at line 50 of file ObservationAggregator.cxx.

48.31.3 Member Function Documentation

48.31.3.1 template<class time_vector, class observation_vector, class observation_vector2> void
Verdandi::ObservationAggregator::Aggregate ( const time_vector &observation_time, const Vector<double> &contribution, const observation_vector2 &observation, double time, observation_vector &aggregated_observation )

Computes an aggregated observation vector over a list of times.

The observations that have a non-zero contribution at time \textit{time} are aggregated.

Parameters

\begin{tabular}{|c|c|}
\hline
\textbf{in} & \textit{observation\_time} & the times of \textit{observation}. \\
\textbf{in} & \textit{contribution} & the contributions associated with observations. \\
\textbf{in} & \textit{observation} & the observations to be aggregated. \\
\textbf{in} & \textit{time} & the time at which the observations should be aggregated. \\
\textbf{out} & \textit{aggregated\_observation} & the aggregated observation vector. \\
\hline
\end{tabular}

Definition at line 350 of file ObservationAggregator.cxx.
Computes aggregated observation over a list of times and observations.

The observations that have a non-zero contribution at time \( \text{time} \) are aggregated. The variables associated with the new aggregated observations vector are stored in \( \text{aggregated\_variable} \).

### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{time} )</td>
<td>the time of the given observations.</td>
</tr>
<tr>
<td>( \text{contribution} )</td>
<td>the contributions associated with observations.</td>
</tr>
<tr>
<td>( \text{variable} )</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>( \text{observation} )</td>
<td>the given observation observation.</td>
</tr>
<tr>
<td>( \text{time} )</td>
<td>a given time.</td>
</tr>
<tr>
<td>( \text{aggregated_variable} )</td>
<td>the variables associated with the aggregated observations.</td>
</tr>
<tr>
<td>( \text{aggregated_observation} )</td>
<td>the aggregated observation.</td>
</tr>
</tbody>
</table>

Definition at line 393 of file ObservationAggregator.cxx.

Computes aggregated observation over a list of times and observations.

The observations that have a non-zero contribution at time \( \text{time} \) are aggregated. The variables associated with the new aggregated observations vector are stored in \( \text{aggregated\_variable} \). The indexes associated with the new aggregated observations vector are stored in \( \text{aggregated\_index} \).

### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{time} )</td>
<td>the time of the given observations.</td>
</tr>
<tr>
<td>( \text{contribution} )</td>
<td>the contributions associated with observations.</td>
</tr>
<tr>
<td>( \text{variable} )</td>
<td>variables associated with observations.</td>
</tr>
<tr>
<td>( \text{index} )</td>
<td>corresponding observation locations.</td>
</tr>
<tr>
<td>( \text{observation} )</td>
<td>the given observation observation.</td>
</tr>
<tr>
<td>( \text{time} )</td>
<td>a given time.</td>
</tr>
<tr>
<td>( \text{aggregated_variable} )</td>
<td>variables associated with the aggregated observations.</td>
</tr>
<tr>
<td>( \text{aggregated_index} )</td>
<td>the aggregated locations.</td>
</tr>
<tr>
<td>( \text{aggregated_observation} )</td>
<td>the aggregated observation.</td>
</tr>
</tbody>
</table>

Definition at line 473 of file ObservationAggregator.cxx.
Computes the contributions of given observations at a given time.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>the given time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>observation_time</td>
<td>the times associated with the given observations.</td>
</tr>
<tr>
<td>out</td>
<td>contribution</td>
<td>the contributions computed.</td>
</tr>
</tbody>
</table>

Definition at line 596 of file ObservationAggregator.cxx.

Computes the contributions of given observations at a given time.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>the given time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>observation_time</td>
<td>the times associated with the given observations.</td>
</tr>
<tr>
<td>in</td>
<td>width_left</td>
<td>the non constant triangle width left.</td>
</tr>
<tr>
<td>in</td>
<td>width_right</td>
<td>the non constant triangle width right.</td>
</tr>
<tr>
<td>out</td>
<td>contribution</td>
<td>the contributions computed.</td>
</tr>
</tbody>
</table>

Definition at line 655 of file ObservationAggregator.cxx.

Computes the contribution of an observation given a time difference.

Parameters

| in    | delta_t         | the time difference. |

Definition at line 687 of file ObservationAggregator.cxx.

Creates a new track.

Returns

The index of the new track.

Definition at line 500 of file ObservationAggregator.cxx.

Activates or deactivates the option 'discard_observation'.
Parameters

| in | discard_\_observation | if set to true, each observation will be used at most one time. |

Definition at line 110 of file ObservationAggregator.cxx.

48.31.3.9 void Verdandi::ObservationAggregator::GetContributionInterval ( double time, double & time_inf, double & time_sup, int & selection_policy ) const

Returns the contribution time interval corresponding to a given time.

This method returns the time interval into which observations have a non-zero contribution at time \textit{time}. An integer is associated to this interval to indicate the observation selection policy. '0' indicates that all observations available in the given interval have to be considered. '-1' indicates that all observations available in the interval \([time\_inf ; time]\) have to be considered. '1' indicates that all observations available in the interval \([time ; time\_sup]\) have to be considered. '2' indicates that only the closest left observation of the interval from \textit{time} and the closest right observation are requested. '3' indicates that all observation in the given interval have to be considered, but, one should take into account non constant triangle widths.

Parameters

| in | time | a given time. |
| out | time\_inf | lower bound of the time interval. |
| out | time\_sup | upper bound (excluded) of the time interval. |
| out | selection\_policy | interval selection policy. |

Definition at line 138 of file ObservationAggregator.cxx.

48.31.3.10 string Verdandi::ObservationAggregator::GetName ( ) const

Returns the name of the class.

Returns

The name of the class.

Definition at line 730 of file ObservationAggregator.cxx.

48.31.3.11 template< class time\_vector > void Verdandi::ObservationAggregator::GetValueIndex ( time\_vector & X, double value, int & index\_inf, int & index\_sup ) const

Returns the least interval containing a given value.

This method returns the index of the closest value among the vector \textit{X} elements that is lower than the given value \textit{value} and the index of the closest value that is higher.

Parameters

| in | X | a sorted vector. |
| in | value | a given value. |
| out | index\_inf | index of the closest value lower than \textit{value}. |
| out | index\_sup | index of the closest value higher than \textit{value}. |

Definition at line 707 of file ObservationAggregator.cxx.
48.31.3.12  void Verdandi::ObservationAggregator::Initialize ( string configuration_file )

Initializer.

Parameters

| in       | configuration_file | configuration file. |

Definition at line 67 of file ObservationAggregator.cxx.

48.31.3.13  double Verdandi::ObservationAggregator::LastTime ( ) const

Returns the last time of the current track.

Returns

The last time of the current track.

Definition at line 530 of file ObservationAggregator.cxx.

48.31.3.14  double Verdandi::ObservationAggregator::LastTime ( int track ) const

Returns the last time of a given track.

Parameters

| in     | track | a given track. |

Returns

The last time of the given track.

Definition at line 543 of file ObservationAggregator.cxx.

48.31.3.15  void Verdandi::ObservationAggregator::PushTime ( double time )

Pushes a time to the current track.

Parameters

| in     | time | a given time. |

Definition at line 555 of file ObservationAggregator.cxx.

48.31.3.16  void Verdandi::ObservationAggregator::PushTime ( double time, int track )

Pushes a time to a given track.

Parameters

| in     | track | index of a given track. |
| in     | time  | a given time. |

Definition at line 568 of file ObservationAggregator.cxx.
Sets the active track to a given track.

Parameters

| in | track | the given track. |

Definition at line 514 of file ObservationAggregator.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/observation_manager/ObservationAggregator.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/observation_manager/ObservationAggregator.cxx

This class simply performs a forward simulation.

#include <ObservationGenerator.hxx>

Inheritance diagram for Verdandi::ObservationGenerator:

```
Verdandi::ObservationGenerator
Verdandi::VerdandiBase
```

Public Types

- typedef Model::state model_state
  Type of the model state vector.
- typedef ObservationManager::observation observation
  Type of the observation vector.

Public Member Functions

- ObservationGenerator ()
  Main constructor.
- ~ObservationGenerator ()
  Destructor.
- void Initialize (string configuration_file, bool initialize_model=true, bool initialize_observation_manager=true)
  Initializes the simulation.
- void Initialize (VerdandiOps &configuration, bool initialize_model=true, bool initialize_observation_manager=true)
  Initializes the simulation.
- void InitializeStep ()
  Initializes the model before a time step.
- void Forward ()
  Performs a step forward without optimal interpolation.
- void FinalizeStep ()
  ```
Finalizes a step for the model.

- void **Finalize** ()
  Finalizes the model.

- bool **HasFinished** ()
  Checks whether the model has finished.

- Model & **GetModel** ()
  Returns the model.

- OutputSaver & **GetOutputSaver** ()
  Returns the output saver.

- string **GetName** () const
  Returns the name of the class.

- void **Message** (string message)
  Receives and handles a message.

Static Public Member Functions

- static void **StaticMessage** (void *object, string message)
  Receives and handles a message with a static method.

Protected Attributes

- Model **model_**
  Underlying model.

- ObservationManager **observation_manager_**
  Observation manager.

- int **iteration_**
  Iteration.

- Vector<double> **time_**
  Time vector.

- string **configuration_file_**
  Path to the configuration file.

- string **model_configuration_file_**
  Path to the model configuration file.

- string **observation_configuration_file_**
  Path to the configuration file for the observation manager.

- map< string, bool > **option_display_**
  Display options.

- OutputSaver **output_saver_**
  Output saver.

48.32.1 Detailed Description

This class simply performs a forward simulation.
48.32.2 Constructor & Destructor Documentation

48.32.2.1 Verdandi::ObservationGenerator::ObservationGenerator ( )

Main constructor.
Builds the driver and reads option keys in the configuration file.

Parameters

| in | configuration | configuration file. |

Definition at line 43 of file ObservationGenerator.cxx.

48.32.3 Member Function Documentation

48.32.3.1 Model & Verdandi::ObservationGenerator::GetModel ( )

Returns the model.

Returns

The model.

Definition at line 337 of file ObservationGenerator.cxx.

48.32.3.2 string Verdandi::ObservationGenerator::GetName ( ) const [virtual]

Returns the name of the class.

Returns

The name of the class.

Reimplemented from Verdandi::VerdandiBase.
Definition at line 360 of file ObservationGenerator.cxx.

48.32.3.3 OutputSaver & Verdandi::ObservationGenerator::GetOutputSaver ( )

Returns the output saver.

Returns

The output saver.

Definition at line 349 of file ObservationGenerator.cxx.

48.32.3.4 bool Verdandi::ObservationGenerator::HasFinished ( )

Checks whether the model has finished.

Returns

True if no more data assimilation is required, false otherwise.

Definition at line 326 of file ObservationGenerator.cxx.
48.32.3.5 void Verdandi::ObservationGenerator::Initialize ( string configuration_file, bool initialize_model = true, bool initialize_observation_manager = true )

Initializes the simulation.
Initializes the model.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration_file</td>
<td>configuration file to be given to the model initialization method.</td>
</tr>
</tbody>
</table>

Definition at line 77 of file ObservationGenerator.hxx.

48.32.3.6 void Verdandi::ObservationGenerator::Initialize ( VerdandiOps & configuration, bool initialize_model = true, bool initialize_observation_manager = true )

Initializes the simulation.
Initializes the model.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>configuration file to be given to the model initialization method.</td>
</tr>
</tbody>
</table>

Definition at line 92 of file ObservationGenerator.hxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/ObservationGenerator.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/ObservationGenerator.cxx

48.33 Verdandi::ObservationManagerTemplate Class Reference

This class is a template of observation manager.

#include <ObservationManagerTemplate.hxx>

Inheritance diagram for Verdandi::ObservationManagerTemplate:

```
Verdandi::VerdandiBase

Verdandi::ObservationManagerTemplate
```

Public Types

- typedef Matrix< double > tangent_linear_operator
  Type of the tangent linear operator.
- typedef Vector< double > tangent_linear_operator_row
  Type of a row of the tangent linear operator.
- typedef Matrix< double > error_variance
  Type of the observation error covariance matrix.
- typedef Vector< double > observation
  Type of the observation vector.
Public Member Functions

- **ObservationManagerTemplate ()**
  Default constructor.
- **∼ObservationManagerTemplate ()**
  Destructor.
- **template<class Model>
  void Initialize (Model &model, string configuration_file)**
  Initializes the observation manager.
- **void DiscardObservation (bool discard_observation)**
  Activates or deactivates the option ‘discard_observations’.
- **template<class Model>
  void SetTime (Model &model, double time)**
  Sets the time of observations to be loaded.
- **observation & GetObservation ()**
  Returns the observations.
- **template<class state>
  observation & GetInnovation (const state &x)**
  Returns an innovation.
- **bool HasObservation () const**
  Indicates if some observations are available at current time.
- **int GetNobservation () const**
  Returns the number of available observations.
- **template<class state, class mat>
  void GetNudgingMatrix (const state &x, mat &M) const**
- **template<class state>
  void ApplyOperator (const state &x, observation &y) const**
  Applies the observation operator to a given vector.
- **template<class state>
  void ApplyTangentLinearOperator (const state &x, observation &y) const**
  Applies the tangent linear operator to a given vector.
- **double GetTangentLinearOperator (int i, int j) const**
  Returns an element of the tangent linear operator.
- **tangent_linear_operator_row & GetTangentLinearOperatorRow (int row) const**
  Returns a row of the tangent linear operator.
- **const tangent_linear_operator& GetTangentLinearOperator () const**
  Returns the tangent linear operator.
- **template<class state>
  void ApplyAdjointOperator (const state &x, observation &y) const**
  Applies the adjoint operator to a given vector.
- **double GetErrorVariance (int i, int j) const**
  Returns an observation error covariance.
- **const error_variance & GetErrorVariance () const**
  Returns the observation error variance.
- **const error_variance & GetErrorVarianceInverse () const**
  Returns the inverse of the observation error covariance matrix.
- **string GetName () const**
  Returns the name of the class.
- **void Message (string message)**
  Receives and handles a message.
Static Public Member Functions

- static void **StaticMessage**(void *object, string message)
  
  Receives and handles a message with a static method.

48.33.1 Detailed Description

This class is a template of observation manager.

48.33.2 Member Function Documentation

48.33.2.1 **template<class state> void Verdandi::ObservationManagerTemplate::ApplyAdjointOperator ( const state & x, observation & y ) const**

Applies the adjoint operator to a given vector.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> x</td>
</tr>
<tr>
<td><strong>out</strong> y</td>
</tr>
</tbody>
</table>

Definition at line 293 of file ObservationManagerTemplate.cxx.

48.33.2.2 **template<class state> void Verdandi::ObservationManagerTemplate::ApplyOperator ( const state & x, observation & y ) const**

Applies the observation operator to a given vector.

This method is called after ‘SetTime’ set the time at which the operator is defined.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> x</td>
</tr>
<tr>
<td><strong>out</strong> y</td>
</tr>
</tbody>
</table>

Definition at line 214 of file ObservationManagerTemplate.cxx.

48.33.2.3 **template<class state> void Verdandi::ObservationManagerTemplate::ApplyTangentLinearOperator ( const state & x, observation & y ) const**

Applies the tangent linear operator to a given vector.

This method is called after ‘SetTime’ set the time at which the operator is defined.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> x</td>
</tr>
<tr>
<td><strong>out</strong> y</td>
</tr>
</tbody>
</table>

Definition at line 231 of file ObservationManagerTemplate.cxx.

48.33.2.4 **void Verdandi::ObservationManagerTemplate::DiscardObservation ( bool discard_observation )**

Activates or deactivates the option ‘discard_observation’.
Parameters

| in | discard_observation | if set to true, each observation will be used at most one time. |

Definition at line 80 of file ObservationManagerTemplate.cxx.

48.33.2.5 double Verdandi::ObservationManagerTemplate::GetErrorVariance ( int i, int j ) const

Return an observation error covariance.

Parameters

| in | i | row index. |
| in | j | column index. |

Returns

The element \((i, j)\) of the observation error variance.

Definition at line 307 of file ObservationManagerTemplate.cxx.

48.33.2.6 const ObservationManagerTemplate::error_variance & Verdandi::ObservationManagerTemplate::GetErrorVariance ( ) const

Returns the observation error variance.

Returns

The observation error covariance matrix.

Definition at line 320 of file ObservationManagerTemplate.cxx.

48.33.2.7 const ObservationManagerTemplate::error_variance & Verdandi::ObservationManagerTemplate::GetErrorVarianceInverse ( ) const

Returns the inverse of the observation error covariance matrix.

Returns

The inverse of the matrix of the observation error covariance.

Definition at line 334 of file ObservationManagerTemplate.cxx.

48.33.2.8 template< class state > ObservationManagerTemplate::observation & Verdandi::ObservationManagerTemplate::GetInnovation ( const state & x )

Returns an innovation.

This method is called after 'SetTime' set the time at which the innovation is requested.

Parameters

| in | state | state vector. |
Returns
The innovation vector.

Definition at line 153 of file ObservationManagerTemplate.cxx.

48.33.2.9 string Verdandi::ObservationManagerTemplate::GetName ( ) const [virtual]

Returns the name of the class.

Returns
The name of the class.

Reimplemented from Verdandi::VerdandiBase.

Definition at line 347 of file ObservationManagerTemplate.cxx.

48.33.2.10 int Verdandi::ObservationManagerTemplate::GetNobservation ( ) const

Returns the number of available observations.

Returns
The total number of observation at current time.

Definition at line 178 of file ObservationManagerTemplate.cxx.

48.33.2.11 ObservationManagerTemplate::observation & Verdandi::ObservationManagerTemplate::GetObservation ( )

Returns the observations.
This method is called after ‘SetTime’ set the time at which the observations are requested.

Returns
The observation vector.

Definition at line 132 of file ObservationManagerTemplate.cxx.

48.33.2.12 double Verdandi::ObservationManagerTemplate::GetTangentLinearOperator ( int i, int j ) const

Returns an element of the tangent linear operator.
This method is called after ‘SetTime’ set the time at which the operator is defined.

Parameters
<table>
<thead>
<tr>
<th>in</th>
<th>i</th>
<th>row index.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>j</td>
<td>column index.</td>
</tr>
</tbody>
</table>

Returns
The element $(i, j)$ of the tangent linear operator.

Definition at line 246 of file ObservationManagerTemplate.cxx.
48.33.2.13 const ObservationManagerTemplate::tangent_linear_operator& Verdandi::ObservationManagerTemplate- ::GetTangentLinearOperator ( ) const

Returns the tangent linear operator.
This method is called after ‘SetTime’ set the time at which the operator is defined.

Returns

The matrix of the tangent linear operator.

Definition at line 276 of file ObservationManagerTemplate.cxx.

48.33.2.14 ObservationManagerTemplate::tangent_linear_operator_row & Verdandi::ObservationManagerTemplate- ::GetTangentLinearOperatorRow ( int row )

Returns a row of the tangent linear operator.
This method is called after ‘SetTime’ set the time at which the operator is defined.

Parameters

| in | row | row index. |

Returns

The row row of the tangent linear operator.

Definition at line 262 of file ObservationManagerTemplate.cxx.

48.33.2.15 template < class Model > void Verdandi::ObservationManagerTemplate::Initialize ( Model & model, string configuration_file )

Initializes the observation manager.

Parameters

| in | model | model. |

| in | configuration_file | configuration file. |

Template Parameters

Model the model type; e.g. ShallowWater<double>

Definition at line 66 of file ObservationManagerTemplate.cxx.

48.33.2.16 template < class Model > void Verdandi::ObservationManagerTemplate::SetTime ( Model & model, double time )

Sets the time of observations to be loaded.

Parameters

| in | model | the model. |

| in | time | a given time. |

Definition at line 95 of file ObservationManagerTemplate.cxx.

The documentation for this class was generated from the following files:
This class performs optimal interpolation.

```cpp
#include <OptimalInterpolation.hxx>
```

Inheritance diagram for Verdandi::OptimalInterpolation:

```
Verdandi::OptimalInterpolation
Verdandi::VerdandiBase
```

Public Types

- `typedef Model::state_error_variance_row model_state_error_variance_row`
  
  Type of a row of the background error variance.

- `typedef Model::state model_state`
  
  Type of the model state vector.

- `typedef Model::state::value_type Ts`
  
  Value type of the model state vector.

- `typedef ObservationManager::observation::value_type To`
  
  Value type of the observations.

- `typedef Model::matrix_state_observation matrix_state_observation`
  
  Value type of the model parameters.

- `typedef ObservationManager::tangent_linear_operator observation_tangent_linear_operator`
  
  Type of the tangent linear observation operator.

- `typedef ObservationManager::tangent_linear_operator_row observation_tangent_linear_operator_row`
  
  Type of a row of the tangent linear observation operator.

- `typedef ObservationManager::observation observation`
  
  Type of the observation vector.

Public Member Functions

- `OptimalInterpolation ()`
  
  Main constructor.

- `~OptimalInterpolation ()`
  
  Destructor.

- `void Initialize (string configuration_file, bool initialize_model=true, bool initialize_observation_manager=true)`
  
  Initializes the optimal interpolation driver.

- `void Initialize (VerdandiOps &configuration, bool initialize_model=true, bool initialize_observation_manager=true)`
  
  Initializes the optimal interpolation driver.

- `void InitializeStep ()`
Initializes a step for the optimal interpolation.
- void **Forward** ()
  Performs a step forward, with optimal interpolation at the end.
- void **Prediction** ()
  Performs a forecast step.
- void **Analyze** ()
  Computes an analysis.
- void **FinalizeStep** ()
  Finalizes a step for the model.
- void **Finalize** ()
  Finalizes the model.
- bool **HasFinished** ()
  Checks whether the model has finished.
- Model & **GetModel** ()
  Returns the model.
- ObservationManager & **GetObservationManager** ()
  Returns the observation manager.
- OutputSaver & **GetOutputSaver** ()
  Returns the output saver.
- string **GetName** () const
  Returns the name of the class.
- void **Message** (string message)
  Receives and handles a message.

**Static Public Member Functions**

- static void **StaticMessage** (void *object, string message)
  Receives and handles a message with a static method.

**Protected Attributes**

- Model **model_**
  Underlying model.
- ObservationManager **observation_manager_**
  Observation manager.
- int **iteration_**
  Iteration.
- string **configuration_file_**
  Path to the configuration file.
- string **model_configuration_file_**
  Path to the model configuration file.
- string **observation_configuration_file_**
  Path to the configuration file for the observation manager.
- map<string, bool> **option_display_**
  Display options.
- int **Nstate_**
  Dimension of the state.
- int **Nobservation_**
  Number of observations.
- bool **analyze_first_step_**
Should an analysis be computed at the first step?

- **string** `blue_computation_`
  
  Computation mode for BLUE: "vector" or "matrix".

- **bool** `with_analysis_variance_diagonal_`
  
  Should the diagonal of the analysis variance be computed?

- **model_state** `analysis_variance_diagonal_`
  
  Diagonal of the analysis variance.

- **OutputSaver** `output_saver_`
  
  Output saver.

48.34.1 Detailed Description

This class performs optimal interpolation.

48.34.2 Member Typedef Documentation

48.34.2.1 typedef Model::matrix state, observation Verdandi::OptimalInterpolation::matrix_state_observation

Value type of the model parameters.

Type of the model/observation crossed matrix.

Definition at line 54 of file OptimalInterpolation.hxx.

48.34.3 Constructor & Destructor Documentation

48.34.3.1 Verdandi::OptimalInterpolation::OptimalInterpolation ( )

Main constructor.

Builds the driver and reads option keys in the configuration file.

Parameters

| in | configuration_file | configuration file. |

Definition at line 45 of file OptimalInterpolation.cxx.

48.34.4 Member Function Documentation

48.34.4.1 void Verdandi::OptimalInterpolation::Analyze ( )

Computes an analysis.

Whenever observations are available, it computes BLUE.

Definition at line 305 of file OptimalInterpolation.cxx.

48.34.4.2 Model & Verdandi::OptimalInterpolation::GetModel ( )

Returns the model.
Returns
    The model.

Definition at line 415 of file OptimalInterpolation.cxx.

48.34.4.3  string Verdandi::OptimalInterpolation::GetName ( ) const  [virtual]

Returns the name of the class.

Returns
    The name of the class.

Reimplemented from Verdandi::VerdandiBase.

Definition at line 450 of file OptimalInterpolation.cxx.

48.34.4.4  ObservationManager & Verdandi::OptimalInterpolation::GetObservationManager ( )

Returns the observation manager.

Returns
    The observation manager.

Definition at line 427 of file OptimalInterpolation.cxx.

48.34.4.5  OutputSaver & Verdandi::OptimalInterpolation::GetOutputSaver ( )

Returns the output saver.

Returns
    The output saver.

Definition at line 439 of file OptimalInterpolation.cxx.

48.34.4.6  bool Verdandi::OptimalInterpolation::HasFinished ( )

Checks whether the model has finished.

Returns
    True if no more data assimilation is required, false otherwise.

Definition at line 404 of file OptimalInterpolation.cxx.

48.34.4.7  void Verdandi::OptimalInterpolation::Initialize ( string configuration_file, bool initialize_model = true, bool initialize_observation_manager = true )

Initializes the optimal interpolation driver.

Initializes the model and the observation manager. Optionally computes the analysis of the first step.

Definition at line 90 of file OptimalInterpolation.cxx.
48.34.4.8 void Verdandi::OptimalInterpolation::Initialize (VerdandiOps &configuration, bool initialize_model=true, bool initialize_observation_manager=true)

Initializes the optimal interpolation driver.
Initializes the model and the observation manager. Optionally computes the analysis of the first step.
Definition at line 104 of file OptimalInterpolation.hxx.

48.34.4.9 void Verdandi::OptimalInterpolation::InitializeStep()

Initializes a step for the optimal interpolation.
Initializes a step for the model.
Definition at line 240 of file OptimalInterpolation.hxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/OptimalInterpolation.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/OptimalInterpolation.hxx

48.35 Verdandi::OutputSaver Class Reference

This class provides convenient methods to save variables on disk.

#include <OutputSaver.hxx>

Public Member Functions

- OutputSaver()
  Default constructor.
- OutputSaver(string configuration_file, string method_name)
  Main constructor.
- void Initialize(string configuration_file, string method_name)
  Initializes the output saver with a configuration file.
- void Initialize(VerdandiOps &configuration)
  Initializes the output saver with a configuration.
- void Activate()
  Activates the Logger.
- void Deactivate()
  Deactivates the Logger.
- ~OutputSaver()
  Destructor.
- string GetName() const
  Returns the name of the class.
- bool IsSaved(string variable_name) const
  Checks whether a variable is saved by the output saver.
- template<class S> void Save(const S &x, double time, string variable_name)
- template<class S> void Save(const S &x, string variable_name)
- void WriteText(const double &x, string file_name) const
  Writes x of type double in a text file.
48.35 Verdandi::OutputSaver Class Reference

- template<class S>
  void WriteText (const S &x, string file_name) const
- template<class S>
  void WriteBinary (const S &x, string file_name) const
- void WriteBinary (const double &x, string file_name) const

  Writes \( x \) in a binary file.
- template<class T, class Prop, class Allocator>
  void WriteBinary (const Matrix<T, Prop, RowSparse, Allocator> &x, string file_name) const
- template<class T, class Prop, class Allocator>
  void WriteBinary (const Matrix<T, Prop, ColSparse, Allocator> &x, string file_name) const
- template<class S>
  void Empty (string variable_name)

  Empties the output file associated with a variable.
- void Empty (string variable_name)
- void Empty ()

  Empties the output files of all registered variables.
- bool IsVariable (string variable_name) const

  Checks if \( \text{variable\_name} \) is a listed variable.
- void DisplayVariableList () const

  Displays the variables parameters.
- template<>
  void WriteBinary (const float &x, string file_name) const

  Writes \( x \) in a binary file.
- template<>
  void WriteBinary (const int &x, string file_name) const

  Writes \( x \) in a binary file.

48.35.1 Detailed Description

This class provides convenient methods to save variables on disk.

48.35.2 Constructor & Destructor Documentation

48.35.2.1 Verdandi::OutputSaver::OutputSaver ( string configuration_file, string section_name )

Main constructor.
Reads the configuration.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>configuration_file</th>
<th>the configuration file.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>section_name</td>
<td>the section in configuration_file where the configuration is to be read.</td>
</tr>
</tbody>
</table>

Definition at line 52 of file OutputSaver.cxx.

48.35.3 Member Function Documentation

48.35.3.1 template<class S> void Verdandi::OutputSaver::Empty ( string variable_name )

Empties the output file associated with a variable.
Parameters

| in | variable_name | the name of the variable whose file should be emptied. |

Definition at line 233 of file OutputSaver.cxx.

48.35.3.2 string Verdandi::OutputSaver::GetName ( ) const

Returns the name of the class.

Returns

The name of the class, that is, "output saver".

Definition at line 167 of file OutputSaver.cxx.

48.35.3.3 void Verdandi::OutputSaver::Initialize ( string configuration_file, string section_name )

Initializes the output saver with a configuration file.

Reads the configuration.

Parameters

| in | configuration_file | the configuration file. |
| in | section_name       | the section in configuration_file where the configuration is to be read. |

Definition at line 65 of file OutputSaver.cxx.

48.35.3.4 void Verdandi::OutputSaver::Initialize ( VerdandiOps & configuration )

Initializes the output saver with a configuration.

Reads the configuration.

Parameters

| in | configuration | VerdandiOps instance with the configuration. The prefix of configuration should already be set so that all entries are accessible ("mode", "variable_list", "file", ...). |

Definition at line 80 of file OutputSaver.cxx.

48.35.3.5 bool Verdandi::OutputSaver::IsSaved ( string variable_name ) const

Checks whether a variable is saved by the output saver.

Parameters

| in | variable_name | name of the variable to be saved. |

Returns

True if the variable is saved by the output saver, and false if the variable is unknown.

Definition at line 184 of file OutputSaver.cxx.
48.35.3.6 bool Verdandi::OutputSaver::IsVariable ( string variable_name ) const

Checks if variable_name is a listed variable.

Parameters

| in | variable_name | name of the variable to be searched. |

Definition at line 275 of file OutputSaver.cxx.

48.35.3.7 void Verdandi::OutputSaver::WriteBinary ( const double & x, string file_name ) const

Writes x in a binary file.

Parameters

| in | x | variable to be written. |

| in | file_name | output filename. |

Definition at line 341 of file OutputSaver.cxx.

48.35.3.8 template<> void Verdandi::OutputSaver::WriteBinary ( const float & x, string file_name ) const

Writes x in a binary file.

Parameters

| in | x | variable to be written. |

| in | file_name | output filename. |

Definition at line 304 of file OutputSaver.cxx.

48.35.3.9 template<> void Verdandi::OutputSaver::WriteBinary ( const int & x, string file_name ) const

Writes x in a binary file.

Parameters

| in | x | variable to be written. |

| in | file_name | output filename. |

Definition at line 323 of file OutputSaver.cxx.

48.35.3.10 void Verdandi::OutputSaver::WriteText ( const double & x, string file_name ) const

Writes x of type double in a text file.

Parameters

| in | x | double to be written. |

| in | file_name | output filename. |

Definition at line 195 of file OutputSaver.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/share/OutputSaver.hxx
This class is a clamped-bar model.
#include <PetscClampedBar.hxx>

Inheritance diagram for Verdandi::PetscClampedBar:

```
+------------------
|                  |
|                  |
|                  |
| +----------------
| |                |
| |                |
| +----------------
|                  |
```

**Public Types**

- typedef T value_type
  The numerical type (e.g., double).
- typedef T * pointer
  Pointer to the numerical type.
- typedef const T * const_pointer
  Const pointer to the numerical type.
- typedef T & reference
  Reference to the numerical type.
- typedef const T & const_reference
  Const reference to the numerical type.
- typedef Matrix<T, General, PETScMPIDense> state_error_variance
  Type of the background error covariance matrix.
- typedef Matrix<T, General, RowMajor, MallocAlloc<T> > state_error_variance_reduced
  Type of the reduced matrix $U$ in the $LU^TL$ decomposition of the background error covariance matrix.
- typedef Vector<T, PETScPar> state_error_variance_row
  Type of a row of the background error variance.
- typedef Matrix<T, General, PETScMPIDense> matrix_state_observation
  Type of the model/observation crossed matrix.
- typedef Matrix<T, General, PETScMPIDense> tangent_linear_operator
  Type of the tangent linear operator.
- typedef Vector<T, PETScPar> state
  Type of the model state vector.
- typedef Vector<T > parameter
  Type of the model state vector.
- typedef Vector< parameter, Collection > parameter_collection
  Collection of vector state.

**Public Member Functions**

- PetscClampedBar ()
  Constructor.
- PetscClampedBar (string configuration_file)
  Constructor.
Destructor.

- `void Initialize (string configuration_file)`
  Initializes the model.

- `void SetMPICommunicator (MPI_Comm &mpi_communicator)`
  Sets the MPI communicator.

- `void Finalize ()`
  Finalizes the model.

- `void InitializeFirstStep ()`
  Initializes the first time step for the model.

- `void InitializeStep ()`
  Initializes the current time step for the model.

- `void InitializeAdjoint ()`
  Initializes the adjoint model.

- `void Forward (bool update_force=true)`
  Advances one step forward in time.

- `bool HasFinished () const`
  Checks whether the model has finished.

- `void Save ()`
  Saves the simulated data.

- `void BackwardAdjoint (state &state_innovation)`
  Performs one step backward in adjoint model.

- `double ApplyOperator (state &x, bool preserve_state=true, bool update_force=true)`
  Applies the model to a given state vector.

- `double ApplyTangentLinearOperator (state &x)`
  Applies the tangent linear model to a given vector.

- `tangent_linear_operator& GetTangentLinearOperator ()`
  Gets the matrix of the tangent linear model.

- `double GetTime () const`
  Returns the current time.

- `void SetTime (double time)`
  Sets the time of the model to a given time.

- `int GetNstate () const`
  Returns the state vector size.

- `int GetLocalNstate () const`
  Returns the state vector local size.

- `void GetStateCopy (state &state)`
  Provides the reduced state vector.

- `void SetStateCopy (state &state)`
  Sets the reduced state vector.

- `state & GetState ()`
  Provides the reduced state vector.

- `void StateUpdated ()`
  Performs some calculations when the update of the model state is done.

- `state & GetStateLowerBound ()`
  Provides the state lower bound.

- `state & GetStateUpperBound ()`
  Provides the state upper bound.

- `state & GetFullState ()`
  Provides the full state vector.

- `void FullStateUpdated ()`
Performs some calculations when the update of the model state is done.

- **state & GetAdjointState ()**
  Returns the adjoint state vector.

- **void AdjointStateUpdated ()**
- **state_error_variance_row & GetStateErrorVarianceRow (int row)**
  Computes a row of the background error covariance matrix B.

- **state_error_variance & GetStateErrorVariance ()**
  Returns the background error covariance matrix (B).

- **const state_error_variance & GetStateErrorVariance () const**
  Returns the background error covariance matrix (B).

- **state_error_variance & GetStateErrorVarianceProjector ()**
- **state_error_variance_reduced & GetStateErrorVarianceReduced ()**
- **const state_error_variance & GetStateErrorVarianceInverse () const**
  Returns the inverse of the background error variance (B\(^{-1}\)).

- **string GetName () const**
  Returns the name of the class.

- **void Message (string message)**
  Receives and handles a message.

### Static Public Member Functions

- **static void StaticMessage (void *object, string message)**
  Receives and handles a message with a static method.

### Protected Attributes

- **double bar_length_**
  Bar length.

- **double Delta_x_**
  Space step along x.

- **int Nx_**
  Number of elements along x.

- **int Ndof_**
  Number of degrees of freedom (dofs).

- **int Nstate_**
  Size of the state vector.

- **int Nreduced_**
  Size of the reduced state vector.

- **double Delta_t_**
  Time step.

- **double time_**
  Current time.

- **double final_time_**
  Simulation duration.

- **double mass_density_**
  Mass parameter.

- **double Young_modulus_**
  Young's Modulus.

- **parameter theta_force_**
  Force parameter.
int Ntheta_force_
   Number of force parameter regions.
Vector<int> theta_force_index_
   Force parameter region of elements.

parameter theta_stiffness_
   Stiffness parameter.
int Ntheta_stiffness_
   Number of stiffness parameter regions.
Vector<int> theta_stiffness_index_
   Stiffness parameter region of elements.

parameter theta_damp_
   Damp parameter.
int Ntheta_damp_
   Number of damp parameter regions.
Vector<int> theta_damp_index_
   Damp parameter region of elements.

parameter theta_mass_
   Mass parameter.
int Ntheta_mass_
   Number of mass parameter regions.
Vector<int> theta_mass_index_
   Mass parameter region of elements.

parameter_collection parameter_
   Parameter collection.
set<string> stable_
   State.
vector<string> reduced_
   Reduced state.
state displacement_0_
   Displacement.
state velocity_0_
   Velocity.
state rhs_
   Force.
int Nstate_local_
   Local size of state vector.
state state_
   Petsc state.
Matrix<T, General, RowMajor> mass_FEM_matrix_
Matrix<T, General, RowMajor> stiffness_FEM_matrix_
Matrix<T, General, RowMajor> damp_FEM_matrix_
Matrix<T, General, PETScMPIAIJ> mass_
Mass matrix.
Matrix<T, General, PETScMPIAIJ> newmark_0_
   Newmark matrix 0.
bool newmark_0_assembled_
Matrix<T, General, PETScMPIAIJ> newmark_1_
   Newmark matrix 1.
bool newmark_1_assembled_
Matrix<T, General, PETScMPIAIJ> damp_
   Damp matrix.
– Matrix<T, General, PETScMPIAIJ> stiffness_
  
  Stiffness matrix.
– double alpha_
  
  Damp alpha coefficient.
– double beta_
  
  Damp beta coefficient.
– double state_error_variance_value_
  
  Background error variance.
– state_error_variance state_error_variance_
  
  Background error covariance matrix (B).
– state_error_variance state_error_variance_inverse_
  
  Inverse of the background error covariance matrix (B^-1).
– state_error_variance state_error_variance_projector_
  
  Projector matrix L in the decomposition of the background error covariance matrix (B) as a product LUL^T.
– state_error_variance_reduced state_error_variance_reduced_
  
  Reduced matrix U in the decomposition of the background error covariance matrix (B) as a product LUL^T.
– bool variance_projector_allocated_
  
  Is state error variance projector allocated?
– bool variance_reduced_allocated_
  
  Is reduced state error variance allocated?
– int current_row_
  
  Index of the row of B currently stored.
– state_error_variance_row state_error_variance_row_
  
  Value of the row of B currently stored.
– int world_rank_
  
  The rank of the current process in MPI_COMM_WORLD.
– int rank_
  
  The rank of the current process in mpi_communicator_.
– int Nworld_process_
  
  The number of processes in MPI_COMM_WORLD.
– int Nprocess_
  
  The number of processes in mpi_communicator_.
– MPI_Comm mpi_communicator_
  
  MPI communicator.
– OutputSaver output_saver_
  
  Output saver.

Static Protected Attributes

– static const double Pi_ = 3.141592653589793238462

48.36.1 Detailed Description

This class is a clamped-bar model.

Template Parameters

\[ T \] the type of floating-point numbers.
48.36.2 Constructor & Destructor Documentation

48.36.2.1 Verdandi::PetscClampedBar::PetscClampedBar ( string configuration_file )

Constructor.
It builds allocates the state vectors.

Parameters

|    | configuration_file | path to the configuration file. |

Definition at line 60 of file PetscClampedBar.cxx.

48.36.3 Member Function Documentation

48.36.3.1 void Verdandi::PetscClampedBar::AdjointStateUpdated ( )

Performs some calculations when the update of the adjoint state is done.
Definition at line 666 of file PetscClampedBar.cxx.

48.36.3.2 double Verdandi::PetscClampedBar::ApplyOperator ( state & x, bool preserve_state = true, bool update_force = true )

Applies the model to a given state vector.

Parameters

|    | x | on entry, the state vector to which the model is applied; on exit, the state vector after the model is applied. |
|    | preserve_state | Boolean to indicate if the model state has to be preserved. |

Returns
The time associated with x on exit plus one time step.

Warning
The time of the model has to be preserved.
Definition at line 381 of file PetscClampedBar.cxx.

48.36.3.3 double Verdandi::PetscClampedBar::ApplyTangentLinearOperator ( state & increment_state )

Applies the tangent linear model to a given vector.

Parameters

|    | increment | the increment. |

Returns
The time associated with x on exit. This time should be the model time plus one time step.
Definition at line 420 of file PetscClampedBar.cxx.
48.36.3.4 void Verdandi::PetscClampedBar::BackwardAdjoint ( state & observation_term )

Performs one step backward in adjoint model.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>observation_term</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H^T R^{-1} (y - Hx)$</td>
<td></td>
</tr>
</tbody>
</table>

Definition at line 360 of file PetscClampedBar.cxx.

48.36.3.5 PetscClampedBar< T >::state & Verdandi::PetscClampedBar::GetAdjointState ( )

Returns the adjoint state vector.

Returns

The adjoint state vector.

Definition at line 656 of file PetscClampedBar.cxx.

48.36.3.6 PetscClampedBar< T >::state & Verdandi::PetscClampedBar::GetFullState ( )

Provides the full state vector.

Returns

state the full state vector.

Definition at line 636 of file PetscClampedBar.cxx.

48.36.3.7 int Verdandi::PetscClampedBar::GetLocalNstate ( ) const

Returns the state vector local size.

Returns

The state vector local size.

Definition at line 483 of file PetscClampedBar.cxx.

48.36.3.8 int Verdandi::PetscClampedBar::GetNstate ( ) const

Returns the state vector size.

Returns

The state vector size.

Definition at line 472 of file PetscClampedBar.cxx.

48.36.3.9 PetscClampedBar< T >::state & Verdandi::PetscClampedBar::GetState ( )

Provides the reduced state vector.

Returns

state the reduced state vector.

Definition at line 555 of file PetscClampedBar.cxx.
48.36.3.10 void Verdandi::PetscClampedBar::GetStateCopy ( state & x )

Provides the reduced state vector.

Parameters

| out | state | the reduced state vector. |

Definition at line 495 of file PetscClampedBar.cxx.

48.36.3.11 PetscClampedBar< T >::state_error_variance & Verdandi::PetscClampedBar::GetStateErrorVariance ( )

Returns the background error covariance matrix ( $B$).

Returns

The matrix of the background error covariance.

Definition at line 697 of file PetscClampedBar.cxx.

48.36.3.12 const PetscClampedBar< T >::state_error_variance & Verdandi::PetscClampedBar::GetStateErrorVariance ( ) const

Returns the background error covariance matrix ( $B$).

Returns

The matrix of the background error covariance.

Definition at line 709 of file PetscClampedBar.cxx.

48.36.3.13 const PetscClampedBar< T >::state_error_variance & Verdandi::PetscClampedBar::GetStateErrorVarianceInverse ( ) const

Returns the inverse of the background error variance ( $B^{-1}$).

Returns

The inverse of the background error variance ( $B^{-1}$).

Definition at line 778 of file PetscClampedBar.cxx.

48.36.3.14 PetscClampedBar< T >::state_error_variance & Verdandi::PetscClampedBar::GetStateErrorVarianceProjector ( )

Returns the matrix $L$ in the decomposition of the state error covariance matrix ( $B$) as a product $LU L^T$.

Returns

The matrix $L$.

Definition at line 723 of file PetscClampedBar.cxx.
Returns the matrix $U$ in the decomposition of the state error covariance matrix ($B$) as a product $LU^T$.

Returns
The matrix $U$.

Definition at line 753 of file PetscClampedBar.cxx.

Computes a row of the background error covariance matrix $B$.

Parameters

<table>
<thead>
<tr>
<th>IN</th>
<th>row</th>
<th>row index</th>
</tr>
</thead>
</table>

Returns
The value of row number $row$.

Definition at line 679 of file PetscClampedBar.cxx.

Provides the state lower bound.

Returns
The state lower bound (componentwise).

Definition at line 612 of file PetscClampedBar.cxx.

Provides the state upper bound.

Returns
The state upper bound (componentwise).

Definition at line 624 of file PetscClampedBar.cxx.

Gets the matrix of the tangent linear model.

Returns
The matrix of the tangent linear model.

Definition at line 433 of file PetscClampedBar.cxx.
48.36.3.20  double Verdandi::PetscClampedBar::GetTime ( ) const

Returns the current time.

Returns

The current time.

Definition at line 450 of file PetscClampedBar.cxx.

48.36.3.21  bool Verdandi::PetscClampedBar::HasFinished ( ) const

Checks whether the model has finished.

Returns

True if no more data assimilation is required, false otherwise.

Definition at line 338 of file PetscClampedBar.cxx.

48.36.3.22  void Verdandi::PetscClampedBar::Initialize ( string configuration_file )

Initializes the model.

Parameters

| in | configuration_file | configuration file. |

Definition at line 84 of file PetscClampedBar.cxx.

48.36.3.23  void Verdandi::PetscClampedBar::Save ( )

Saves the simulated data.

It saves the displacement 'disp_0_' and the velocity 'velo_0_'.

Definition at line 348 of file PetscClampedBar.cxx.

48.36.3.24  void Verdandi::PetscClampedBar::SetMPICommunicator ( MPI_Comm & mpi_communicator )

Sets the MPI communicator.

Parameters

| in                | mpi::communicator | the MPI communicator to be set. |

Definition at line 232 of file PetscClampedBar.cxx.

48.36.3.25  void Verdandi::PetscClampedBar::SetStateCopy ( state & x )

Sets the reduced state vector.

Before setting the reduced state vector, special requirements can be enforced; e.g. positivity requirement or inferior and superior limits.
Parameters

| in   | state | the reduced state vector. |

Definition at line 524 of file PetscClampedBar.cxx.

48.36.3.26 void Verdandi::PetscClampedBar::SetTime ( double time )

Sets the time of the model to a given time.

Parameters

| in   | time | a given time. |

Definition at line 461 of file PetscClampedBar.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/model/PetscClampedBar.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/model/PetscClampedBar.cxx

48.37 Verdandi::PetscLinearObservationManager Class Reference

Linear observation operator.

#include <PetscLinearObservationManager.hxx>

Inheritance diagram for Verdandi::PetscLinearObservationManager:

```
Verdandi::VerdandiBase
```

```
Verdandi::PetscLinearObservationManager
```

Public Types

- typedef Matrix< T, General, PETScMPIAIJ > tangent_linear_operator
- typedef Matrix< T > error_variance
  
  Type of the observation error covariance matrix.
- typedef Vector< T > tangent_linear_operator_row
  
  Type of a row of the tangent linear operator.
- typedef Vector< T > observation
  
  Type of the observation vector.
- typedef Vector< T > observation_vector
  
  Type of the observation vector.
- typedef Vector2< T > observation_vector2
  
  Type of the observation vector 2.
- typedef Vector3< T > observation_vector3
  
  Type of the observation vector 3.
- typedef Vector< int > variable_vector
  
  Type of the variable vector.
- typedef Vector2< int > variable_vector2
  
  Type of the variable vector 2.
- typedef Vector3< int > variable_vector3
  Type of the variable vector 3.
- typedef Vector< int > index_vector
  Type of the index vector.
- typedef Vector2< int > index_vector2
  Type of the index vector 2.
- typedef Vector3< int > index_vector3
  Type of the index vector 3.
- typedef Vector< double > time_vector
  Type of the time vector.
- typedef Vector2< double > time_vector2
  Type of the time vector 2.
- typedef Vector3< double > time_vector3
  Type of the time vector 3.

Public Member Functions

- PetscLinearObservationManager ()
  Default constructor.
- template<class Model>
  PetscLinearObservationManager (Model &model, string configuration_file)
  Main constructor.
- ~PetscLinearObservationManager ()
  Destructor.
- template<class Model>
  void Initialize (Model &model, string configuration_file)
  Initializes the observation manager.
- template<class Model>
  void InitializeOperator (Model &model, string configuration_file)
  Initializes the observation manager.
- void SetMPICommunicator (MPI_Comm &mpi_communicator)
  Sets the MPI communicator.
- void DiscardObservation (bool discard_observation)
  Activates or deactivates the option 'discard_observation'.
- int CreateTrack ()
  Creates a new track.
- void SetTrack (int track)
  Sets the track to a given track.
- template<class Model>
  void SetTime (Model &model, double time)
  Sets the time of observations to be loaded.
- void SetTime (double time)
  Sets the time of observations to be loaded.
- void SetAvailableTime (double time, time_vector &available_time)
  Sets the available observation times at a given time.
- void SetAvailableTime (double time_inf, double time_sup, time_vector &available_time)
  Sets available observation times at a given time interval.
- void SetAvailableTime (double time, double time_inf, double time_sup, int selection_policy, time_vector &available_time)
  Sets available observation times at a given time interval.
- void GetFlattenedObservation (double time, observation_vector &observation)
- void GetFlattenedObservation (double time_inf, double time_sup, observation_vector &observation)
  Gets observations flattened over a list of times.
- void GetFlattenedObservation (observation_vector &observation)
  Gets observations flattened over a list of times.
- void GetFlattenedObservation (const time_vector &available_time, observation_vector &observation)
  Gets observations flattened over a list of times.
- void GetFlattenedObservation (double time, variable_vector &observation_variable, observation_vector &observation)
  Gets observations flattened over a list of times.
- void GetFlattenedObservation (double time_inf, double time_sup, variable_vector &observation_variable, observation_vector &observation)
  Gets observations flattened over a list of times.
- void GetFlattenedObservation (variable_vector &observation_variable, observation_vector &observation)
  Gets observations flattened over a list of times.
- void GetFlattenedObservation (const time_vector &available_time, variable_vector &observation_variable, observation_vector &observation)
  Gets observations flattened over a list of times.
- void GetFlattenedObservation (double time, variable_vector &observation_variable, index_vector &observation_index, observation_vector &observation)
  Gets observations flattened over a list of times.
- void GetFlattenedObservation (double time_inf, double time_sup, variable_vector &observation_variable, index_vector &observation_index, observation_vector &observation)
  Gets observations flattened over a list of times.
- void GetFlattenedObservation (variable_vector &observation_variable, index_vector &observation_index, observation_vector &observation)
  Gets observations flattened over a list of times.
- void GetFlattenedObservation (const time_vector &available_time, variable_vector &observation_variable, index_vector &observation_index, observation_vector &observation)
  Gets observations flattened over a list of times.
- void GetAggregatedObservation (double time, observation_vector &observation)
  Gets observations aggregated over a list of times.
- void GetAggregatedObservation (double time_inf, double time_sup, observation_vector &observation)
  Gets observations aggregated over a list of times.
- void GetAggregatedObservation (observation_vector &observation)
  Gets observations aggregated over a list of times.
- void GetAggregatedObservation (const time_vector &available_time, observation_vector &observation)
  Gets observations aggregated over a list of times.
- void GetAggregatedObservation (double time, variable_vector &observation_variable, observation_vector2 &observation2)
  Gets observations aggregated over a list of times.
- void GetAggregatedObservation (double time_inf, double time_sup, variable_vector &observation_variable, observation_vector2 &observation2)
  Gets observations aggregated over a list of times.
- void GetAggregatedObservation (variable_vector &observation_variable, observation_vector2 &observation2)
  Gets observations aggregated over a list of times.
- void GetAggregatedObservation (const time_vector &available_time, variable_vector &observation_variable, observation_vector2 &observation2)
  Gets observations aggregated over a list of times.
- void GetAggregatedObservation (double time, variable_vector &observation_variable, index_vector2 &observation_index2, observation_vector2 &observation2)
  Gets observations aggregated over a list of times.
Gets observations aggregated over a list of times.
- void GetAggregatedObservation (double time_inf, double time_sup, variable_vector &observation_variable, index_vector2 &observation_index2, observation_vector2 &observation2)
  Gets observations aggregated over a list of times.
- void GetAggregatedObservation (variable_vector &observation_variable, index_vector2 &observation_index2, observation_vector2 &observation2)
  Gets observations aggregated over a list of times.
- void GetAggregatedObservation (const time_vector &available_time, variable_vector &observation_variable, index_vector2 &observation_index2, observation_vector2 &observation2)
  Gets observations aggregated over a list of times.
- void GetRawObservation (double time, observation_vector2 &observation2)
  Gets available observations at a given time.
- void GetRawObservation (double time_inf, double time_sup, observation_vector2 &observation2)
  Gets observations available in a given interval.
- void GetRawObservation (observation_vector2 &observation2)
  Gets observations at the current time.
- void GetRawObservation (const time_vector &available_time, observation_vector2 &observation2)
  Gets observations of a list of times.
- void GetRawObservation (double time, variable_vector2 &observation_variable2, observation_vector3 &observation3)
  Gets available observations at a given time.
- void GetRawObservation (double time_inf, double time_sup, variable_vector2 &observation_variable2, observation_vector3 &observation3)
  Gets observations available in a given interval.
- void GetRawObservation (variable_vector2 &observation_variable2, observation_vector3 &observation3)
  Gets observations at the current time.
- void GetRawObservation (const time_vector &available_time, variable_vector2 &observation_variable2, observation_vector3 &observation3)
  Gets observations of a list of times.
- void GetRawObservation (double time, variable_vector2 &observation_variable2, index_vector3 &observation_index3, observation_vector3 &observation3)
  Gets available observations at a given time.
- void GetRawObservation (double time_inf, double time_sup, variable_vector2 &observation_variable2, index_vector3 &observation_index3, observation_vector3 &observation3)
  Gets observations available in a given interval.
- void GetRawObservation (variable_vector2 &observation_variable2, index_vector3 &observation_index3, observation_vector3 &observation3)
  Gets observations at the current time.
- void GetRawObservation (const time_vector &available_time, variable_vector2 &observation_variable2, index_vector3 &observation_index3, observation_vector3 &observation3)
  Gets observations of a list of times.
- void ReadObservationVariable (const time_vector &available_time, variable_vector2 &observation_variable2) const
  Builds variables vector associated with given observations.
- void ReadObservation (const time_vector &available_time, const variable_vector2 &observation_variable2, observation_vector3 &observation3) const
  Builds observations associated with given times and variables.
- void ReadObservation (const time_vector &available_time, observation_vector2 &observation2) const
  Builds observations associated with given times.
- void ReadObservation (ifstream &file_stream, double time, int variable, observation_vector &observation) const
  Reads observation from observation file given a time and a variable.
Class Documentation

- void ReadObservationIndex (const time_vector &available_time, const variable_vector2 &observation_variable2, index_vector3 &observation_index3) const
  Reads observations indexes.
- void ReadObservationTriangleWidth (double time_inf, double time_sup, Vector<double> &width_left, Vector<double> &width_right) const
  Reads triangle width associated with observations of a given interval.
- observation & GetObservation ()
  Gets observation.
- template<class state >
  observation & GetInnovation (const state &x)
  Gets innovation.
- bool HasObservation () const
  Indicates if some observations are available at current time.
- int GetNobservation () const
  Gets Nobservation_ value.
- bool IsOperatorSparse () const
  Checks whether the observation operator is available in a sparse matrix.
- bool IsErrorSparse () const
  Checks whether the observation error covariance matrix is sparse.
- bool HasErrorMatrix () const
  Checks whether the observation error covariance is available in a matrix.
- template<class state >
  void ApplyOperator (const state &x, observation &y) const
  Applies the operator to a given vector.
- template<class state >
  void ApplyTangentLinearOperator (const state &x, observation &y) const
  Applies the tangent linear operator to a given vector.
- T GetTangentLinearOperator (int i, int j) const
  Linearized observation operator.
- tangent_linear_operator_row & GetTangentLinearOperatorRow (int row)
  Linearized observation operator.
- const tangent_linear_operator& GetTangentLinearOperator () const
  Linearized observation operator.
- template<class state >
  void ApplyAdjointOperator (const state &x, observation &y) const
  Applies the adjoint operator to a given vector.
- bool HasBLUECorrection () const
  Checks whether a BLUE correction is available.
- void GetBLUECorrection (Vector<T> &BLUE_correction) const
  Gets the BLUE correction.
- T GetErrorVariance (int i, int j) const
  Observation error covariance.
- const error_variance & GetErrorVariance () const
  Observation error covariance matrix.
- const error_variance & GetErrorVarianceInverse () const
  Inverse of the observation error covariance matrix.
- string GetName () const
  Returns the name of the class.
- void Message (string message)
  Receives and handles a message.
Static Public Member Functions

- static void StaticMessage (void *object, string message)
  Receives and handles a message with a static method.

Protected Attributes

- string observation_file_
  File that stores the observations.
- string observation_type_
  How are stored the observations.
- int Nobservation_
  Total number of observations at current time.
- size_t Nbyte_observation_
  Size in bytes of an observation vector.
- double Delta_t_
  Period with which observations are available.
- int Nskip_
  Period with which available observations are actually loaded.
- double initial_time_
  First time at which observations are available.
- double final_time_
  Final time at which observations are available.
- observation current_observation_
  Current observation.
- bool observation_loaded_
  Are current observations loaded.
- int world_rank_
  The rank of the current process in MPI_COMM_WORLD.
- int rank_
  The rank of the current process in mpi_communicator_.
- int Nworld_process_
  The number of processes in MPI_COMM_WORLD.
- int Nprocess_
  The number of processes in mpi_communicator_.
- MPI_Comm mpi_communicator_
  MPI communicator.
- double time_
  Requested time.
- time_vector available_time_
  Available observation time of the time interval.
- Vector<double> contribution_
  Contribution associated with available observations.
- OBSERVATION_AGGREGATOR<T> observation_aggregator_
  Observations aggregator.
- tangent_linear_operator tangent_operator_matrix_
  Tangent operator matrix (H).
- bool operator_scaled_identity_
  Is the operator a scaled identity matrix?
- T operator_diagonal_value_
  In case of a scaled identity operator.
– T  \texttt{error\_variance\_value} 
  \textit{Observation error variance.}

– \texttt{error\_variance error\_variance\_inverse} 
  \textit{Inverse of the observation error covariance matrix (R).}

– string \texttt{width\_file} 
  \textit{File that stores the observations.}

– int \texttt{Nstate\_model} 
  \textit{The size of a model state vector.}

– int \texttt{Nlocal\_state\_model} 
  \textit{The size of a local model state vector.}

– \texttt{observation\_innovation} 
  \textit{Innovation currently stored.}

– int \texttt{current\_row} 
  \textit{Index of the row of H currently stored.}

– \texttt{tangent\_linear\_operator\_row tangent\_operator\_row} 
  \textit{Value of the row of H currently stored.}

\subsection{48.37.1 Detailed Description}
Linear observation operator.

\textbf{Template Parameters}

\begin{verbatim}
T  \textit{the type of floating-point numbers.}
\end{verbatim}

\subsection{48.37.2 Constructor & Destructor Documentation}

\subsubsection{48.37.2.1 Verdandi::PetscLinearObservationManager::PetscLinearObservationManager ( )}

Default constructor.

It entirely defines the operator: no dimension or size is associated with this implementation.

Definition at line 45 of file PetscLinearObservationManager.cxx.

\subsubsection{48.37.2.2 template\langle class Model \rangle Verdandi::PetscLinearObservationManager::PetscLinearObservationManager ( Model \& model, string configuration\_file )

Main constructor.

\textbf{Parameters}

\begin{verbatim}
in  \texttt{model}  \textit{model.}
in  \texttt{configuration\_file}  \textit{configuration\_file.}
\end{verbatim}

\textbf{Template Parameters}

\begin{verbatim}
Model  \textit{the model type; e.g. ShallowWater\langle double \rangle}
\end{verbatim}

Definition at line 60 of file PetscLinearObservationManager.cxx.

\subsection{48.37.3 Member Function Documentation}
48.37.3.1 template<class state> void Verdandi::PetscLinearObservationManager::ApplyAdjointOperator ( const state & x, observation & y ) const

Applies the adjoint operator to a given vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x a vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>y the value of the operator at x.</td>
</tr>
</tbody>
</table>

Definition at line 1685 of file PetscLinearObservationManager.cxx.

48.37.3.2 template<class state> void Verdandi::PetscLinearObservationManager::ApplyOperator ( const state & x, observation & y ) const

Applies the operator to a given vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x a vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>y the value of the operator at x.</td>
</tr>
</tbody>
</table>

Definition at line 1601 of file PetscLinearObservationManager.cxx.

48.37.3.3 template<class state> void Verdandi::PetscLinearObservationManager::ApplyTangentLinearOperator ( const state x, observation & y ) const

Applies the tangent linear operator to a given vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x a vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>y the value of the tangent linear operator at x.</td>
</tr>
</tbody>
</table>

Definition at line 1626 of file PetscLinearObservationManager.cxx.

48.37.3.4 int Verdandi::PetscLinearObservationManager::CreateTrack ( )

Creates a new track.

Returns

The index of the new track.

Definition at line 235 of file PetscLinearObservationManager.cxx.

48.37.3.5 void Verdandi::PetscLinearObservationManager::DiscardObservation ( bool discard_observation )

Activates or deactivates the option ‘discard_observation’.

Parameters

| in  | discard_observation if set to true, each observation will be used at most one time. |

Definition at line 223 of file PetscLinearObservationManager.cxx.
48.37.3.6  void Verdandi::PetscLinearObservationManager::GetAggregatedObservation ( double time, observation_vector & observation )

Gets observations aggregated over a list of times. The observations available at the given time are loaded and aggregated.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>the given time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation</td>
<td>the aggregated observations.</td>
</tr>
</tbody>
</table>

Definition at line 737 of file PetscLinearObservationManager.cxx.

48.37.3.7  void Verdandi::PetscLinearObservationManager::GetAggregatedObservation ( double time_inf, double time_sup, observation_vector & observation )

Gets observations aggregated over a list of times. The observations in the interval \([time\_inf, time\_sup]\) are loaded and aggregated.

Parameters

| in   | time\_inf | lower bound of the given interval. |
|------| time\_sup | upper bound of the given interval. |
| out  | observation | the aggregated observations. |

Definition at line 756 of file PetscLinearObservationManager.cxx.

48.37.3.8  void Verdandi::PetscLinearObservationManager::GetAggregatedObservation ( observation_vector & observation )

Gets observations aggregated over a list of times. The observations available are loaded and aggregated.

Parameters

| out  | observation | the aggregated observations. |

Definition at line 773 of file PetscLinearObservationManager.cxx.

48.37.3.9  void Verdandi::PetscLinearObservationManager::GetAggregatedObservation ( const time_vector & available_time, observation_vector & observation )

Gets observations aggregated over a list of times. The observations available at the times \(available\_time\) are loaded and aggregated.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>available_time</th>
<th>the given observation time vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation</td>
<td>the aggregated observations.</td>
</tr>
</tbody>
</table>

Definition at line 787 of file PetscLinearObservationManager.cxx.
void Verdandi::PetscLinearObservationManager::GetAggregatedObservation ( double time, variable_vector & observation_variable, observation_vector2 & observation2 )

Gets observations aggregated over a list of times.
The observations available at the given time are loaded and aggregated.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>the given time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation_variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation2</td>
<td>the aggregated observations.</td>
</tr>
</tbody>
</table>

Definition at line 810 of file PetscLinearObservationManager.cxx.

void Verdandi::PetscLinearObservationManager::GetAggregatedObservation ( double time_inf, double time_sup, variable_vector & observation_variable, observation_vector2 & observation2 )

Gets observations aggregated over a list of times.
The observations in the interval $[\text{time}_{inf}, \text{time}_{sup}]$ are loaded and aggregated.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time_inf</th>
<th>lower_bound of the given interval.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>time_sup</td>
<td>upper_bound of the given interval.</td>
</tr>
<tr>
<td>out</td>
<td>observation_variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation2</td>
<td>the aggregated observations.</td>
</tr>
</tbody>
</table>

Definition at line 833 of file PetscLinearObservationManager.cxx.

void Verdandi::PetscLinearObservationManager::GetAggregatedObservation ( variable_vector & observation_variable, observation_vector2 & observation2 )

Gets observations aggregated over a list of times.
The observations available are loaded and aggregated.

Parameters

| out   | observation_variable | variables associated with the observations. |
| out   | observation2 | the aggregated observations. |

Definition at line 854 of file PetscLinearObservationManager.cxx.

void Verdandi::PetscLinearObservationManager::GetAggregatedObservation ( const time_vector & available_time, variable_vector & observation_variable, observation_vector2 & observation2 )

Gets observations aggregated over a list of times.
The observations available at the times available_time are loaded and aggregated.
### Parameters

| in | available_time | the given observation time vector. |
| out | observation_variable | variables associated with the observations. |
| out | observation2 | the aggregated observations. |

Definition at line 872 of file PetscLinearObservationManager.cxx.

48.37.3.14 void Verdandi::PetscLinearObservationManager::GetAggregatedObservation ( double time, variable_vector & observation_variable, index_vector2 & observation_index2, observation_vector2 & observation2 )

Gets observations aggregated over a list of times.
The observations available at the given time are loaded and aggregated.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
</tr>
<tr>
<td>out</td>
</tr>
<tr>
<td>out</td>
</tr>
<tr>
<td>out</td>
</tr>
</tbody>
</table>

Definition at line 904 of file PetscLinearObservationManager.cxx.

48.37.3.15 void Verdandi::PetscLinearObservationManager::GetAggregatedObservation ( double time_inf, double time_sup, variable_vector & observation_variable, index_vector2 & observation_index2, observation_vector2 & observation2 )

Gets observations aggregated over a list of times.
The observations in the interval \([time\_inf, time\_sup]\) are loaded and aggregated.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
</tr>
<tr>
<td>in</td>
</tr>
<tr>
<td>out</td>
</tr>
<tr>
<td>out</td>
</tr>
<tr>
<td>out</td>
</tr>
</tbody>
</table>

Definition at line 929 of file PetscLinearObservationManager.cxx.

48.37.3.16 void Verdandi::PetscLinearObservationManager::GetAggregatedObservation ( variable_vector & observation_variable, index_vector2 & observation_index2, observation_vector2 & observation2 )

Gets observations aggregated over a list of times.
The observations available are loaded and aggregated.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
</tr>
</tbody>
</table>
### 48.37.3.17 void Verdandi::PetscLinearObservationManager::GetAggregatedObservation (const time_vector & available_time, variable_vector & observation_variable, index_vector2 & observation_index2, observation_vector2 & observation2)

Gets observations aggregated over a list of times.

The observations available at the times `available_time` are loaded and aggregated.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>available_time</th>
<th>the given observation time vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation_variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation_index2</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation2</td>
<td>the aggregated observations.</td>
</tr>
</tbody>
</table>

Definition at line 973 of file PetscLinearObservationManager.cxx.

### 48.37.3.18 void Verdandi::PetscLinearObservationManager::GetBLUECorrection (Vector<T> & BLUE_correction) const

Gets the BLUE correction.

**Parameters**

| out | BLUE_correction | BLUE correction vector. |

Definition at line 1712 of file PetscLinearObservationManager.cxx.

### 48.37.3.19 T Verdandi:: PetscLinearObservationManager::GetErrorVariance (int i, int j) const

Observation error covariance.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>i</th>
<th>row index.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>j</td>
<td>column index.</td>
</tr>
</tbody>
</table>

**Returns**

The element \((i, j)\) of the observation error covariance.

Definition at line 1726 of file PetscLinearObservationManager.cxx.

### 48.37.3.20 const PetscLinearObservationManager<T> & Verdandi:: PetscLinearObservationManager::GetErrorVariance ( ) const

Observation error covariance matrix.
Returns

The matrix of the observation error covariance.

Definition at line 1742 of file PetscLinearObservationManager.cpp.

48.37.3.21 const PetscLinearObservationManager< T >::error_variance & Verdandi::PetscLinearObservationManager::GetErrorVarianceInverse ( ) const

Inverse of the observation error covariance matrix.

Returns

Inverse of the matrix of the observation error covariance.

Definition at line 1756 of file PetscLinearObservationManager.cpp.

48.37.3.22 void Verdandi::PetscLinearObservationManager::GetFlattenedObservation ( double time, observation_vector & observation )

Gets observations flattened over a list of times.

The observations available at time \( \text{time} \) are loaded and concatenated in a vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>\text{time}</th>
<th>the given time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>\text{observation}</td>
<td>the observation to be flattened.</td>
</tr>
</tbody>
</table>

Definition at line 484 of file PetscLinearObservationManager.cpp.

48.37.3.23 void Verdandi::PetscLinearObservationManager::GetFlattenedObservation ( double time_inf, double time_sup, observation_vector & observation )

Gets observations flattened over a list of times.

The observations in the interval \([\text{time}_{\text{inf}}, \text{time}_{\text{sup}}]\) are loaded and concatenated in a vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>\text{time}_{\text{inf}}</th>
<th>lower bound of the given interval.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>\text{time}_{\text{sup}}</td>
<td>upper bound of the given interval.</td>
</tr>
<tr>
<td>out</td>
<td>\text{observation}</td>
<td>the observation to be flattened.</td>
</tr>
</tbody>
</table>

Definition at line 502 of file PetscLinearObservationManager.cpp.

48.37.3.24 void Verdandi::PetscLinearObservationManager::GetFlattenedObservation ( observation_vector & observation )

Gets observations flattened over a list of times.

The observations available are loaded and concatenated in a vector.

Parameters

| out    | \text{observation} | the observation to be loaded. |

Definition at line 517 of file PetscLinearObservationManager.cpp.
48.37.3.25 void Verdandi::PetscLinearObservationManager::GetFlattenedObservation ( const time_vector & available_time, observation_vector & observation )

Gets observations flattened over a list of times.
The observations available at the times `available_time` are loaded and concatenated in a vector.

**Parameters**

| in | available_time | the given observation time vector. |
| out | observation     | the observation to be loaded. |

Definition at line 531 of file PetscLinearObservationManager.cxx.

48.37.3.26 void Verdandi::PetscLinearObservationManager::GetFlattenedObservation ( double time, variable_vector & observation_variable, observation_vector & observation )

Gets observations flattened over a list of times.
The observations available at time `time` are loaded and concatenated in a vector.

**Parameters**

| in | time | the given time. |
| out | observation_variable | variables associated with the observations. |
| out | observation | the observation to be flattened. |

Definition at line 553 of file PetscLinearObservationManager.cxx.

48.37.3.27 void Verdandi::PetscLinearObservationManager::GetFlattenedObservation ( double time_inf, double time_sup, variable_vector & observation_variable, observation_vector & observation )

Gets observations flattened over a list of times.
The observations in the interval `[time_inf, time_sup]` are loaded and concatenated in a vector.

**Parameters**

| in | time_inf | lower_bound of the given interval. |
| in | time_sup | upper_bound of the given interval. |
| out | observation_variable | variables associated with the observations. |
| out | observation | the observation to be flattened. |

Definition at line 575 of file PetscLinearObservationManager.cxx.

48.37.3.28 void Verdandi::PetscLinearObservationManager::GetFlattenedObservation ( variable_vector & observation_variable, observation_vector & observation )

Gets observations flattened over a list of times.
The observations available are loaded and concatenated in a vector.

**Parameters**

| out | observation_variable | variables associated with the observations. |
| out | observation | the observation to be loaded. |
48.37.3.29 void Verdandi::PetscLinearObservationManager::GetFlattenedObservation ( const time_vector & available_time, variable_vector & observation_variable, observation_vector & observation )

Gets observations flattened over a list of times.
The observations available at the times available_time are loaded and concatenated in a vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>available_time</th>
<th>the given observation time vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation_variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

48.37.3.30 void Verdandi::PetscLinearObservationManager::GetFlattenedObservation ( double time, variable_vector & observation_variable, index_vector & observation_index, observation_vector & observation )

Gets observations flattened over a list of times.
The observations available at time time are loaded and concatenated in a vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>the given time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation_variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation_index</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation</td>
<td>the observation to be flattened.</td>
</tr>
</tbody>
</table>

48.37.3.31 void Verdandi::PetscLinearObservationManager::GetFlattenedObservation ( double time_inf, double time_sup, variable_vector & observation_variable, index_vector & observation_index, observation_vector & observation )

Gets observations flattened over a list of times.
The observations in the interval [time_inf, time_sup] are loaded and concatenated in a vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time_inf</th>
<th>lower_bound of the given interval.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>time_sup</td>
<td>upper_bound of the given interval.</td>
</tr>
<tr>
<td>out</td>
<td>observation_variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation_index</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation</td>
<td>the observation to be flattened.</td>
</tr>
</tbody>
</table>
void Verdandi::PetscLinearObservationManager::GetFlattenedObservation ( variable_vector & observation_variable, index_vector & observation_index, observation_vector & observation )

Gets observations flattened over a list of times.
The observations available are loaded and concatenated in a vector.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>out</strong></td>
<td>observation_variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td><strong>out</strong></td>
<td>observation_index</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td><strong>out</strong></td>
<td>observation</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 684 of file PetscLinearObservationManager.cxx.

void Verdandi::PetscLinearObservationManager::GetFlattenedObservation ( const time_vector & available_time, variable_vector & observation_variable, index_vector & observation_index, observation_vector & observation )

Gets observations flattened over a list of times.
The observations available at the times available_time are loaded and concatenated in a vector.

Parameters

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td>available_time</td>
<td>the given observation time vector.</td>
</tr>
<tr>
<td><strong>out</strong></td>
<td>observation_variable</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td><strong>out</strong></td>
<td>observation_index</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td><strong>out</strong></td>
<td>observation</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 704 of file PetscLinearObservationManager.cxx.

template<class state > PetscLinearObservationManager< T >::observation & Verdandi::PetscLinearObservationManager::GetInnovation ( const state & x )

Gets innovation.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong></td>
<td>state</td>
</tr>
</tbody>
</table>

Returns

The innovation vector.

Definition at line 1502 of file PetscLinearObservationManager.cxx.

string Verdandi::PetscLinearObservationManager::GetName ( ) const [virtual]

Returns the name of the class.
Returns

The name of the class.

Reimplemented from Verdandi::VerdandiBase.

Definition at line 1768 of file PetscLinearObservationManager.cxx.

48.37.3.36 int Verdandi::PetscLinearObservationManager::GetNoobservation ( ) const

Gets Nobservation_ value.

Returns

The total number of observation at current time.

Definition at line 1531 of file PetscLinearObservationManager.cxx.

48.37.3.37 PetscLinearObservationManager< T >::observation & Verdandi::PetscLinearObservationManager::GetObservation ( )

Gets observation.

Returns

The observation vector.

Definition at line 1478 of file PetscLinearObservationManager.cxx.

48.37.3.38 void Verdandi::PetscLinearObservationManager::GetRawObservation ( double time, observation_vector2 & observation2 )

Gets available observations at a given time.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>the given time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation2</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 1011 of file PetscLinearObservationManager.cxx.

48.37.3.39 void Verdandi::PetscLinearObservationManager::GetRawObservation ( double time_inf, double time_sup, observation_vector2 & observation2 )

Gets observations available in a given interval.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time_inf</th>
<th>lower_bound of the given interval.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>time_sup</td>
<td>upper_bound of the given interval.</td>
</tr>
<tr>
<td>out</td>
<td>observation2</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 1029 of file PetscLinearObservationManager.cxx.
48.37.3.40 void Verdandi::PetscLinearObservationManager::GetRawObservation ( observation_vector2 & observation2 )

Gets observations at the current time.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>out</strong></td>
<td><strong>observation2</strong></td>
</tr>
</tbody>
</table>

Definition at line 1045 of file PetscLinearObservationManager.cxx.

48.37.3.41 void Verdandi::PetscLinearObservationManager::GetRawObservation ( const time_vector & available_time, observation_vector2 & observation2 )

Gets observations of a list of times.

Parameters

| **in** | **available_time** | the given observation time vector. |
| **out** | **observation2** | the observation to be loaded. |

Definition at line 1058 of file PetscLinearObservationManager.cxx.

48.37.3.42 void Verdandi::PetscLinearObservationManager::GetRawObservation ( double time, variable_vector2 & observation_variable2, observation_vector3 & observation3 )

Gets available observations at a given time.

Parameters

| **in** | **time** | the given time. |
| **out** | **observation_variable2** | variables associated with the observations. |
| **out** | **observation3** | the observation to be loaded. |

Definition at line 1078 of file PetscLinearObservationManager.cxx.

48.37.3.43 void Verdandi::PetscLinearObservationManager::GetRawObservation ( double time_inf, double time_sup, variable_vector2 & observation_variable2, observation_vector3 & observation3 )

Gets observations available in a given interval.

Parameters

| **in** | **time_inf** | lower_bound of the given interval. |
| **in** | **time_sup** | upper_bound of the given interval. |
| **out** | **observation_variable2** | variables associated with the observations. |
| **out** | **observation3** | the observation to be loaded. |

Definition at line 1100 of file PetscLinearObservationManager.cxx.

48.37.3.44 void Verdandi::PetscLinearObservationManager::GetRawObservation ( variable_vector2 & observation_variable2, observation_vector3 & observation3 )

Gets observations at the current time.
Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>observation_variable2</th>
<th>variables associated with the observations.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation3</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 1121 of file PetscLinearObservationManager.cxx.

48.37.3.45 void Verdandi::PetscLinearObservationManager::GetRawObservation ( const time\_vector & available\_time, variable\_vector2 & observation\_variable2, observation\_vector3 & observation3 )

Gets observations of a list of times.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>available_time</th>
<th>the given observation time vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation_variable2</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation3</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 1139 of file PetscLinearObservationManager.cxx.

48.37.3.46 void Verdandi::PetscLinearObservationManager::GetRawObservation ( double time, variable\_vector2 & observation\_variable2, index\_vector3 & observation\_index3, observation\_vector3 & observation3 )

Gets available observations at a given time.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time</th>
<th>the given time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation_variable2</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation_index3</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation3</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 1162 of file PetscLinearObservationManager.cxx.

48.37.3.47 void Verdandi::PetscLinearObservationManager::GetRawObservation ( double time\_inf, double time\_sup, variable\_vector2 & observation\_variable2, index\_vector3 & observation\_index3, observation\_vector3 & observation3 )

Gets observations available in a given interval.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>time_inf</th>
<th>lower_bound of the given interval.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>time_sup</td>
<td>upper_bound of the given interval.</td>
</tr>
<tr>
<td>out</td>
<td>observation_variable2</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation_index3</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation3</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 1186 of file PetscLinearObservationManager.cxx.
48.37.3.48 void Verdandi::PetscLinearObservationManager::GetRawObservation ( variable_vector2 & observation_variable2, index_vector3 & observation_index3, observation_vector3 & observation3 )

Gets observations at the current time.

Parameters

<table>
<thead>
<tr>
<th>out</th>
<th>observation__variable2</th>
<th>variables associated with the observations.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation__index3</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation3</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 1209 of file PetscLinearObservationManager.cxx.

48.37.3.49 void Verdandi::PetscLinearObservationManager::GetRawObservation ( const time_vector & available_time, variable_vector2 & observation_variable2, index_vector3 & observation_index3, observation_vector3 & observation3 )

Gets observations of a list of times.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>available_time</th>
<th>the given observation time vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>observation__variable2</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation__index3</td>
<td>indexes associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation3</td>
<td>the observation to be loaded.</td>
</tr>
</tbody>
</table>

Definition at line 1229 of file PetscLinearObservationManager.cxx.

48.37.3.50 T Verdandi::PetscLinearObservationManager::GetTangentLinearOperator ( int i, int j ) const

Linearized observation operator.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>i</th>
<th>row index.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>j</td>
<td>column index.</td>
</tr>
</tbody>
</table>

Returns

The element \((i, j)\) of the linearized operator.

Definition at line 1642 of file PetscLinearObservationManager.cxx.

48.37.3.51 const PetscLinearObservationManager< T >::tangent_linear_operator& Verdandi::PetscLinearObservationManager::GetTangentLinearOperator ( ) const

Linearized observation operator.

Returns

The matrix of the linearized operator.

Definition at line 1670 of file PetscLinearObservationManager.cxx.
### Class Documentation

**PetscLinearObservationManager**

#### `template < T > ::tangent_linear_operator_row & Verdandi::PetscLinearObservationManager::GetTangentLinearOperatorRow ( int row )`

Linearized observation operator.

**Parameters**

| in | row | row index. |

**Returns**

The row `row` of the linearized operator.

Definition at line 1656 of file PetscLinearObservationManager.cxx.

#### `bool Verdandi::PetscLinearObservationManager::HasBLUECorrection ( ) const`

Checks whether a BLUE correction is available.

**Returns**

True if a BLUE correction is available, false otherwise.

Definition at line 1699 of file PetscLinearObservationManager.cxx.

#### `bool Verdandi::PetscLinearObservationManager::HasErrorMatrix ( ) const`

Checks whether the observation error covariance is available in a matrix.

**Returns**

True if the observation error covariance is available in a matrix, false otherwise.

Definition at line 1578 of file PetscLinearObservationManager.cxx.

#### `template < class Model > void Verdandi::PetscLinearObservationManager::Initialize ( Model & model, string configuration_file )`

Initializes the observation manager.

**Parameters**

| in | model | model. |
| in | configuration_file | configuration file. |

**Template Parameters**

| Model | the model type; e.g. ShallowWater<double> |

Definition at line 90 of file PetscLinearObservationManager.cxx.

#### `template < class Model > void Verdandi::PetscLinearObservationManager::InitializeOperator ( Model & model, string configuration_file )`

Initializes the observation manager.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>model</th>
<th>model.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>configuration_file</td>
<td>configuration file.</td>
</tr>
</tbody>
</table>

Template Parameters

| Model | the model type; e.g. ShallowWater<double> |

Definition at line 156 of file PetscLinearObservationManager.cxx.

48.37.3.57 bool Verdandi::PetscLinearObservationManager::IsErrorSparse ( ) const

Checks whether the observation error covariance matrix is sparse.

Returns

True if the observation error covariance matrix is sparse, false otherwise.

Definition at line 1561 of file PetscLinearObservationManager.cxx.

48.37.3.58 bool Verdandi::PetscLinearObservationManager::IsOperatorSparse ( ) const

Checks whether the observation operator is available in a sparse matrix.

Returns

True if the observation operator is available in a sparse matrix, false otherwise.

Definition at line 1545 of file PetscLinearObservationManager.cxx.

48.37.3.59 void Verdandi::PetscLinearObservationManager::ReadObservation ( const time_vector & available_time, const variable_vector2 & observation_variable2, observation_vector3 & observation3 ) const

Builds observations associated with given times and variables.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>available_time</th>
<th>the given observation time vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>observation_variable2</td>
<td>variables associated with the observations.</td>
</tr>
<tr>
<td>out</td>
<td>observation3</td>
<td>the observations.</td>
</tr>
</tbody>
</table>

Definition at line 1276 of file PetscLinearObservationManager.cxx.

48.37.3.60 void Verdandi::PetscLinearObservationManager::ReadObservation ( const time_vector & available_time, observation_vector2 & observation2 ) const

Builds observations associated with given times.

Parameters

| in   | available_time | the given observation time vector. |
| out  | observation2 | the observations. |

Definition at line 1318 of file PetscLinearObservationManager.cxx.
48.37.3.61 void Verdandi::PetscLinearObservationManager::ReadObservation ( ifstream & file_stream, double time, int variable, observation_vector & observation ) const

Reads observation from observation file given a time and a variable.

Parameters

| in          | file_stream          | the observation file stream. |
| in          | time                 | the time.                     |
| in          | variable             | the variable.                 |
| out         | observation           | the observations.             |

Definition at line 1353 of file PetscLinearObservationManager.cxx.

48.37.3.62 void Verdandi::PetscLinearObservationManager::ReadObservationIndex ( const time_vector & available_time, const variable_vector2 & observation_variable2, index_vector3 & observation_index3 ) const

Reads observations indexes.

Parameters

| in          | available_time       | the available time.          |
| in          | observation_variable2 | variable associated with the observations. |
| out         | observation_index3   | the indexes associated with the observations. |

Definition at line 1387 of file PetscLinearObservationManager.cxx.

48.37.3.63 void Verdandi::PetscLinearObservationManager::ReadObservationTriangleWidth ( double time_inf, double time_sup, Vector< double > & width_left, Vector< double > & width_right ) const

Reads triangle width associated with observations of a given interval.

Parameters

| in          | time_inf            | lower bound of a given interval. |
| in          | time_sup            | upper bound of a given interval. |
| out         | width_left          | left widths associated with observations. |
| out         | width_right         | right widths associated with observations. |

Definition at line 1418 of file PetscLinearObservationManager.cxx.

48.37.3.64 void Verdandi::PetscLinearObservationManager::ReadObservationVariable ( const time_vector & available_time, variable_vector2 & observation_variable2 ) const

Builds variables vector associated with given observations.

Parameters

| in          | available_time       | the given observation time vector. |
| out         | observation_variable2 | variables associated with the observations. |

Definition at line 1255 of file PetscLinearObservationManager.cxx.
Sets the available observation times at a given time.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>time</td>
<td>the given time.</td>
</tr>
<tr>
<td>out</td>
<td>available_time</td>
<td>the available observation times.</td>
</tr>
</tbody>
</table>

Definition at line 295 of file PetscLinearObservationManager.cxx.

Sets available observation times at a given time interval.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>time_inf</td>
<td>lower bound of the given time interval.</td>
</tr>
<tr>
<td>in</td>
<td>time_sup</td>
<td>upper bound of the given time interval.</td>
</tr>
<tr>
<td>in</td>
<td>selection_policy</td>
<td>interval selection policy.</td>
</tr>
<tr>
<td>out</td>
<td>available_time</td>
<td>the available observation times.</td>
</tr>
</tbody>
</table>

Definition at line 321 of file PetscLinearObservationManager.cxx.

Sets available observation times at a given time interval.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>time_inf</td>
<td>lower bound of the given time interval.</td>
</tr>
<tr>
<td>in</td>
<td>time_sup</td>
<td>upper bound of the given time interval.</td>
</tr>
<tr>
<td>in</td>
<td>selection_policy</td>
<td>interval selection policy.</td>
</tr>
<tr>
<td>out</td>
<td>available_time</td>
<td>the available observation times.</td>
</tr>
</tbody>
</table>

Definition at line 347 of file PetscLinearObservationManager.cxx.

Sets the MPI communicator.

**Parameters**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mpi_communicator</td>
<td>the MPI communicator to be set.</td>
</tr>
</tbody>
</table>

Definition at line 205 of file PetscLinearObservationManager.cxx.

Sets the time of observations to be loaded.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>model</th>
<th>the model.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>time</td>
<td>a given time.</td>
</tr>
</tbody>
</table>

Definition at line 261 of file PetscLinearObservationManager.cxx.

48.37.3.70 void Verdandi::PetscLinearObservationManager::SetTime ( double time )

Sets the time of observations to be loaded.

Parameters

| in  | time        | a given time. |

Definition at line 273 of file PetscLinearObservationManager.cxx.

48.37.3.71 void Verdandi::PetscLinearObservationManager::SetTrack ( int track )

Sets the track to a given track.

Parameters

| in  | track       | the given track. |

Definition at line 247 of file PetscLinearObservationManager.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/observation_manager/PetscLinearObservationManager.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/observation_manager/PetscLinearObservationManager.cxx

48.38 Verdandi::PythonModel Class Reference

Inheritance diagram for Verdandi::PythonModel:

```
Verdandi::VerdandiBase
  ↓
Verdandi::PythonModel
```

Public Types

- typedef Vector< double > state
  Type of the state vector.
- typedef Matrix< double > state_error_variance
  Type of the state error variance.
- typedef Matrix< double > state_error_variance_reduced
  Type of the reduced matrix $U$ in the $LU^T$ decomposition of the background error covariance matrix.
- typedef Vector< double > state_error_variance_row
  Type of a row of the state error variance.
typedef Matrix< double > matrix_state_observation
\textit{Type of the state/observation crossed matrix.}

typedef Matrix< double > tangent_linear_operator
\textit{Type of the tangent linear model.}

typedef Matrix< double > error_variance
\textit{Type of the model error variance.}

typedef Vector< double > uncertain_parameter
\textit{Type of an uncertain parameter.}

\section*{Public Member Functions}

\subsection{Constructor}

\texttt{PythonModel} ()
\textit{Constructor.}

\texttt{PythonModel} (string configuration_file)
\textit{Constructor.}

\texttt{\sim{}PythonModel} ()
\textit{Destructor.}

\texttt{Initialize} (string configuration_file, bool call\_initialize=true)
\textit{Initializes the model.}

\texttt{InitializeStep} ()
\textit{Initializes the current time step for the model.}

\texttt{Forward} ()
\textit{Advances one step forward in time.}

\texttt{BackwardAdjoint} (state &observation\_term)
\textit{Performs one step backward in adjoint model.}

\texttt{HasFinished} () const
\textit{Checks whether the model has finished.}

\texttt{Save} ()
\textit{Saves the simulated data.}

\texttt{FinalizeStep} ()
\textit{Finalizes the current time step for the model.}

\texttt{Finalize} ()
\textit{Finalizes the model.}

\texttt{ApplyOperator} (state &x, bool preserve\_state=true)
\textit{Applies the model to a given vector.}

\texttt{ApplyTangentLinearOperator} (state &x)
\textit{Applies the tangent linear model to a given vector.}

\texttt{tangent\_linear\_operator\& GetTangentLinearOperator} ()
\textit{Gets the tangent linear model.}

\texttt{GetTime} () const
\textit{Returns the current time.}

\texttt{SetTime} (double time)
\textit{Sets the current time.}

\texttt{GetNstate} () const
\textit{Returns the state vector size.}

\texttt{GetNfull\_state} () const
\textit{Returns the full state vector size.}

\texttt{state \& GetState} ()
\textit{Provides the controlled state vector.}

\texttt{StateChanged} ()
Performs some calculations when the update of the model state is done.

- `state & GetStateLowerBound ()`
  Provides the state lower bound.

- `state & GetStateUpperBound ()`
  Provides the state upper bound.

- `state & GetFullState ()`
  Provides the full state vector.

- `void FullStateUpdated ()`
  Performs some calculations when the update of the model state is done.

- `state & GetAdjointState ()`
  Returns the adjoint state vector.

- `void AdjointStateUpdated ()`

- `int GetNparameter ()`
  Returns the number of parameters to be perturbed.

- `uncertain_parameter & GetParameter (int i)`
  Returns the i-th uncertain parameter.

- `void ParameterUpdated (int i)`
  Triggers the necessary actions after an uncertain parameter has been updated.

- `string GetParameterName (int i)`
  Returns the name of a parameter to be perturbed.

- `Vector<double> & GetParameterCorrelation (int i)`
  Returns the correlation between the uncertain parameters.

- `string GetParameterPDF (int i)`
  Returns the PDF of the i-th parameter.

- `Matrix<double, Symmetric, Seldon::RowSymPacked> & GetParameterVariance (int i)`
  Returns the covariance matrix associated with the i-th parameter.

- `Vector<double> & GetParameterPDFData (int i)`
  Returns parameters associated with the PDF of some model parameter.

- `string GetParameterOption (int i)`
  Returns the perturbation option of the i-th parameter.

- `state_error_variance_row & GetStateErrorVarianceRow (int row)`
  Computes a row of the variance of the state error.

- `state_error_variance & GetStateErrorVariance ()`
  Returns the state error variance.

- `state_error_variance & GetStateErrorVarianceProjector ()`

- `state_error_variance_reduced & GetStateErrorVarianceReduced ()`

- `const state_error_variance & GetStateErrorVarianceInverse ()`
  Returns the inverse of the background error variance (B\(^{-1}\)).

- `string GetName () const`
  Returns the name of the model.

- `void Message (string message)`
  Receives and handles a message.

- `PyObject ∗ GetModelInstance ()`
  Returns a pointer to the instance of the Python model.

### Static Public Member Functions

- `static void StaticMessage (void ∗object, string message)`
  Receives and handles a message with a static method.
Protected Attributes

- PyObject * `pyModelInstance_`
  Instance of the Python model class.
- PyObject * `pyModelFile_`
  Name of the python model file.
- PyObject * `pyModelModule_`
  Imported model module from Python.
- PyObject * `pyModelDict_`
  Dictionary object that describes the module’s namespace.
- PyObject * `pyModelClass_`
  Model class from the module.
- bool `is_module_initialized_`
  Has the Python module been initialized?
- string `module_`
  Name of the Python module that contains the model class.
- string `directory_`
  Directory to include in "sys.path" for the module to be found. If no directory is to be added to "sys.path", this attribute remains empty.
- string `class_name_`
  Name of the Python model class.
- Vector<double> `state_`
  State vector.
- Vector<double> `full_state_`
  Full state vector.
- state `upper_bound_`
  State upper bound.
- state `lower_bound_`
  State lower bound.
- state `adjoint_`
  Adjoint state.
- state_error_variance `state_error_variance_`
  State error variance.
- state_error_variance `state_error_variance_inverse_`
  State error variance inverse.
- int `current_row_`
  Number of the row of state_error_variance_ currently stored.
- state_error_variance `row state_error_variance_row_`
  Value of the row of state_error_variance_ currently stored.
- state_error_variance `state_error_variance_projector_`
  Projector matrix L in the decomposition of the background error covariance matrix (B) as a product LUL^T.
- state_error_variance `reduced state_error_variance_reduced_`
  Reduced matrix U in the decomposition of the background error covariance matrix (B) as a product LUL^T.
- tangent_linear_operator `tangent_linear_operator_`
  Tangent linear operator (H).
- uncertain_parameter `uncertain_parameter_`
  Uncertain parameter.
- Vector<double> `parameter_correlation_`
  Correlations between the parameters.
- string `parameter_pdf_`
  Name of the probability distribution.
- Matrix< double, Symmetric, Seldon::RowSymPacked > parameter_variance_
  Covariance matrix.
- Vector< double > parameter_pdf_data_
  PDF parameters.
- string parameter_option_
  Perturbation option.

### 48.38.1 Constructor & Destructor Documentation

#### 48.38.1.1 Verdandi::PythonModel::PythonModel ( string configuration_file )

Constructor.

It reads the initial condition and the time settings.

**Parameters**

| in    | configuration_file | path to the configuration file. |

Definition at line 51 of file PythonModel.cxx.

### 48.38.2 Member Function Documentation

#### 48.38.2.1 void Verdandi::PythonModel::AdjointStateUpdated ( )

 Performs some calculations when the update of the adjoint state is done.

Definition at line 623 of file PythonModel.cxx.

#### 48.38.2.2 double Verdandi::PythonModel::ApplyOperator ( state & x, bool preserve_state = true )

Applies the model to a given vector.

The current state of the model is modified.

**Parameters**

| in,out | x | on entry, the state vector to which the model is applied; on exit, the state vector after the model is applied. |
| in     | preserve_state | Boolean to indicate if the model state has to be preserved. |

**Returns**

The time associated with x on exit. If forward is true, this time should coincide with the model time on exit. If forward is false, this time should coincide with the model time on exit plus one time step.

**Warning**

The time of the model has to be preserved.

Definition at line 268 of file PythonModel.cxx.

#### 48.38.2.3 double Verdandi::PythonModel::ApplyTangentLinearOperator ( state & x )

Applies the tangent linear model to a given vector.
Parameters

| in,out | $x$ on entry, a vector to which the tangent linear model should be applied; on exit, the result. |

Returns

The time associated with $x$ on exit. This time should be the model time plus one time step.

Definition at line 297 of file PythonModel.cxx.

48.38.2.4 void Verdandi::PythonModel::BackwardAdjoint ( state & observation_term )

Performs one step backward in adjoint model.

Parameters

| in   | observation_term $H^T R^{-1} (y - Hx)$ |

Definition at line 184 of file PythonModel.cxx.

48.38.2.5 void Verdandi::PythonModel::Forward ( )

Advances one step forward in time.

$$x_{n+1}^f = \mathcal{M}_h(x^n_a, p_h).$$

Definition at line 169 of file PythonModel.cxx.

48.38.2.6 PythonModel::state & Verdandi::PythonModel::GetAdjointState ( )

Returns the adjoint state vector.

Returns

The adjoint state vector.

Definition at line 591 of file PythonModel.cxx.

48.38.2.7 PythonModel::state & Verdandi::PythonModel::GetFullState ( )

Provides the full state vector.

Returns

The full state vector.

Definition at line 545 of file PythonModel.cxx.

48.38.2.8 string Verdandi::PythonModel::GetName ( ) const [virtual]

Returns the name of the model.
Returns

The name of the model.

Reimplemented from Verdandi::VerdandiBase.
Definition at line 1100 of file PythonModel.hxx.

48.38.2.9 int Verdandi::PythonModel::GetNfull_state ( ) const

Returns the full state vector size.

Returns

The full state vector size.

Definition at line 418 of file PythonModel.hxx.

48.38.2.10 int Verdandi::PythonModel::GetNparameter ( )

Returns the number of parameters to be perturbed.

Returns

The number of parameters to be perturbed.

Definition at line 639 of file PythonModel.hxx.

48.38.2.11 int Verdandi::PythonModel::GetNstate ( ) const

Returns the state vector size.

Returns

The state vector size.

Definition at line 400 of file PythonModel.hxx.

48.38.2.12 PythonModel::uncertain_parameter & Verdandi::PythonModel::GetParameter ( int i )

Returns the i-th uncertain parameter.

Parameters

|  in   | i   | index of the parameter. |

Returns

The vector associated with the i-th parameter.

Definition at line 659 of file PythonModel.hxx.

48.38.2.13 Vector< double > & Verdandi::PythonModel::GetParameterCorrelation ( int i )

Returns the correlation between the uncertain parameters.
Parameters

| in | i | index of the parameter. |

Returns

The correlation between the uncertain parameters.

Definition at line 735 of file PythonModel.cxx.

48.38.2.14 string Verdandi::PythonModel::GetParameterName ( int i )

Returns the name of a parameter to be perturbed.

Parameters

| in | i | index of the parameter. |

Returns

The name of the parameter.

Definition at line 714 of file PythonModel.cxx.

48.38.2.15 string Verdandi::PythonModel::GetParameterOption ( int i )

Returns the perturbation option of the i-th parameter.

Parameters

| in | i | parameter index. |

Returns

The perturbation option of the i-th parameter.

Definition at line 884 of file PythonModel.cxx.

48.38.2.16 string Verdandi::PythonModel::GetParameterPDF ( int i )

Returns the PDF of the i-th parameter.

Parameters

| in | i | uncertain-variable index. |

Returns

The PDF of the i-th parameter.

Definition at line 775 of file PythonModel.cxx.

48.38.2.17 Vector< double > & Verdandi::PythonModel::GetParameterPDFData ( int i )

Returns parameters associated with the PDF of some model parameter.
In case of normal or log-normal distribution, the parameters are clipping parameters.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>in ( i )</td>
</tr>
</tbody>
</table>

**Returns**

The parameters associated with the \( i \)-th parameter.

Definition at line 845 of file PythonModel.cxx.

48.38.2.18  \texttt{Matrix}< \texttt{double}, \texttt{Symmetric}, Seldon::RowSymPacked > & Verdandi::PythonModel::GetParameterVariance ( int \( i \) )

Returns the covariance matrix associated with the \( i \)-th parameter.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>in ( i )</td>
</tr>
</tbody>
</table>

**Returns**

The covariance matrix associated with the \( i \)-th parameter.

Definition at line 797 of file PythonModel.cxx.

48.38.2.19  \texttt{PythonModel::state} & Verdandi::PythonModel::GetState ( )

Provides the controlled state vector.

**Returns**

The controlled state vector.

Definition at line 436 of file PythonModel.cxx.

48.38.2.20  \texttt{PythonModel::state\_error\_variance} & Verdandi::PythonModel::GetStateErrorVariance ( )

Returns the state error variance.

**Returns**

The state error variance.

Definition at line 952 of file PythonModel.cxx.

48.38.2.21  \texttt{const PythonModel::state\_error\_variance} & Verdandi::PythonModel::GetStateErrorVarianceInverse ( )

Returns the inverse of the background error variance \( (B^{-1}) \).

**Returns**

The inverse of the background error variance \( (B^{-1}) \).

Definition at line 1065 of file PythonModel.cxx.
PythonModel::state_error_variance & Verdandi::PythonModel::GetStateErrorVarianceProjector ( )

Returns the matrix \( L \) in the decomposition of the state error covariance matrix (\( B \)) as a product \( LUL^T \).

Returns

The matrix \( L \).

Definition at line 989 of file PythonModel.cxx.

PythonModel::state_error_variance_reduced & Verdandi::PythonModel::GetStateErrorVarianceReduced ( )

Returns the matrix \( U \) in the decomposition of the state error covariance matrix (\( B \)) as a product \( LUL^T \).

Returns

The matrix \( U \).

Definition at line 1028 of file PythonModel.cxx.

PythonModel::state_error_variance_row & Verdandi::PythonModel::GetStateErrorVarianceRow ( int row )

Computes a row of the variance of the state error.

Parameters

| in | row | row index. |

Returns

The row with index row in the state error variance.

Definition at line 911 of file PythonModel.cxx.

PythonModel::state & Verdandi::PythonModel::GetStateLowerBound ( )

Provides the state lower bound.

Returns

The state lower bound (componentwise).

Definition at line 482 of file PythonModel.cxx.

PythonModel::state & Verdandi::PythonModel::GetStateUpperBound ( )

Provides the state upper bound.

Returns

The state upper bound (componentwise).

Definition at line 514 of file PythonModel.cxx.
48.38.2.27  PythonModel::tangent_linear_operator & Verdandi::PythonModel::GetTangentLinearOperator ( )

Gets the tangent linear model.

Returns

The matrix of the tangent linear model.

Definition at line 326 of file PythonModel.cxx.

48.38.2.28  double Verdandi::PythonModel::GetTime ( ) const

Returns the current time.

Returns

The current time.

Definition at line 366 of file PythonModel.cxx.

48.38.2.29  bool Verdandi::PythonModel::HasFinished ( ) const

Checks whether the model has finished.

Returns

True if the simulation is done, false otherwise.

Definition at line 207 of file PythonModel.cxx.

48.38.2.30  void Verdandi::PythonModel::Initialize ( string configuration_file, bool call_initialize = true )

Initializes the model.
It reads the initial condition and the time settings.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>configuration_file</th>
<th>configuration file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>call_initialize</td>
<td>should the 'Initialize' method of the Python object be called?</td>
</tr>
</tbody>
</table>

Definition at line 92 of file PythonModel.cxx.

48.38.2.31  void Verdandi::PythonModel::ParameterUpdated ( int i )

Triggers the necessary actions after an uncertain parameter has been updated.

Parameters

| in | i | index of the parameter. |

Definition at line 696 of file PythonModel.cxx.

48.38.2.32  void Verdandi::PythonModel::SetTime ( double time )

Sets the current time.
Returns

The time step.

Definition at line 384 of file PythonModel.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/model/PythonModel.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/model/PythonModel.cxx

**Public Member Functions**

- def `__init__`:
  Constructs the model.

- def `Initialize`:
  Initializes the model.

- def `InitializeStep`:
  Initializes the current time step for the model.

- def `Forward`:
  Advances one step forward in time.

- def `BackwardAdjoint`:
  Performs one step backward in adjoint model.

- def `ApplyOperator`:
  Applies the model to a given vector.

- def `ApplyTangentLinearOperator`:
  Applies the tangent linear model to a given vector.

- def `GetTangentLinearOperator`:
  Returns the tangent linear model.

- def `HasFinished`:
  Checks whether the model has finished.

- def `FinalizeStep`:
  Finalizes the current time step for the model.

- def `Finalize`:
  Finalizes the model.

- def `GetTime`:
  Returns the current time.

- def `SetTime`:
  Sets the current time.

- def `GetNstate`:
  Returns the dimension of the state.

- def `GetNfull_state`:
  Returns the dimension of the full state.

- def `GetState`:
  Provides the controlled state vector.

- def `StateUpdated`:
  Performs some calculations when the update of the model state is done.
- def GetStateLowerBound
  Provides the state lower bound.
- def GetStateUpperBound
  Provides the state upper bound.
- def GetFullState
  Provides the full state vector.
- def FullStateUpdated
  Performs some calculations when the update of the model state is done.
- def GetAdjointState
  Provides the adjoint state vector.
- def AdjointStateUpdated
  Performs some calculations when the update of the adjoint state is done.
- def GetNparameter
  Returns the number of parameters to be perturbed.
- def GetParameterPDF
  Returns the PDF of the i-th parameter.
- def GetParameter
  Gets the i-th uncertain parameter.
- def ParameterUpdated
  Triggers the actions after an uncertain parameter has been updated.
- def GetParameterName
  Returns the name of a parameter to be perturbed.
- def GetParameterCorrelation
  Returns the correlation between the uncertain parameters.
- def GetParameterPDFData
  Returns parameters associated with the PDF of some model parameter.
- def GetParameterVariance
  Returns the covariance matrix associated with the i-th parameter.
- def GetParameterOption
  Returns the perturbation option of the i-th parameter.
- def GetStateErrorVariance
  Returns the state error variance.
- def GetStateErrorVarianceRow
  Returns a row of the state error variance.
- def GetStateErrorVarianceProjector
  Returns the matrix L in the decomposition of the state error covariance matrix \((B)\) as a product \(LULT\).
- def GetStateErrorVarianceReduced
  Returns the matrix U in the decomposition of the state error covariance matrix \((B)\) as a product \(LULT\).
- def GetStateErrorVarianceInverse
  Returns the inverse of the background error variance \((B^{-1})\).
- def GetName
  Returns the name of the class.
- def Message
  Receives and handles a message.

Public Attributes

- time_
48.39.1 Detailed Description

This class is a template model in Python.

48.39.2 Constructor & Destructor Documentation

48.39.2.1 def PythonModelTemplate::PythonModelTemplate::init ( self )

Constructs the model.
Definition at line 26 of file PythonModelTemplate.py.

48.39.3 Member Function Documentation

48.39.3.1 def PythonModelTemplate::PythonModelTemplate::AdjointStateUpdated ( self )

Performs some calculations when the update of the adjoint state is done.
Definition at line 161 of file PythonModelTemplate.py.

48.39.3.2 def PythonModelTemplate::PythonModelTemplate::ApplyOperator ( self, state, preserve_state = True )

Applies the model to a given vector.
The current state of the model is modified.
Parameters

<table>
<thead>
<tr>
<th>in, out</th>
<th>state</th>
<th>a vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>preserve_state</td>
<td>Boolean to indicate if the model state has to be preserved.</td>
</tr>
</tbody>
</table>

Definition at line 61 of file PythonModelTemplate.py.

48.39.3.3 def PythonModelTemplate::PythonModelTemplate::ApplyTangentLinearOperator ( self, x )

Applies the tangent linear model to a given vector.
Parameters

| in, out | x | on entry, a vector to which the tangent linear model should be applied; on exit, the result. |

Definition at line 68 of file PythonModelTemplate.py.

48.39.3.4 def PythonModelTemplate::PythonModelTemplate::BackwardAdjoint ( self, observation_term )

Performs one step backward in adjoint model.
Parameters

| in | observation_term | $H^T R^{-1} (y - Hx)$ |

Definition at line 52 of file PythonModelTemplate.py.
48.39.3.5 def PythonModelTemplate::PythonModelTemplate::Finalize ( self )

Finalizes the model.
Definition at line 89 of file PythonModelTemplate.py.

48.39.3.6 def PythonModelTemplate::PythonModelTemplate::FinalizeStep ( self )

Finalizes the current time step for the model.
Definition at line 85 of file PythonModelTemplate.py.

48.39.3.7 def PythonModelTemplate::PythonModelTemplate::Forward ( self )

Advances one step forward in time.

\[ x_{h+1}^f = \mathcal{M}_h(x_h^a, p_h) \]

Definition at line 46 of file PythonModelTemplate.py.

48.39.3.8 def PythonModelTemplate::PythonModelTemplate::FullStateUpdated ( self, full_state )

Performs some calculations when the update of the model state is done.
Definition at line 149 of file PythonModelTemplate.py.

48.39.3.9 def PythonModelTemplate::PythonModelTemplate::GetAdjointState ( self )

Provides the adjoint state vector.

Returns
The adjoint state vector.
Definition at line 155 of file PythonModelTemplate.py.

48.39.3.10 def PythonModelTemplate::PythonModelTemplate::GetFullState ( self )

Provides the full state vector.
Returns
The full state vector.
Definition at line 144 of file PythonModelTemplate.py.

48.39.3.11 def PythonModelTemplate::PythonModelTemplate::GetName ( self )

Returns the name of the class.
Returns
The name of the class.
Definition at line 264 of file PythonModelTemplate.py.
48.39.3.12  def PythonModelTemplate::PythonModelTemplate::GetNfull_state ( self )

Returns the dimension of the full state.

Returns
   The dimension of the full state.

Definition at line 116 of file PythonModelTemplate.py.

48.39.3.13  def PythonModelTemplate::PythonModelTemplate::GetNparameter ( self )

Returns the number of parameters to be perturbed.

Returns
   The number of parameters to be perturbed.

Definition at line 167 of file PythonModelTemplate.py.

48.39.3.14  def PythonModelTemplate::PythonModelTemplate::GetNstate ( self )

Returns the dimension of the state.

Returns
   The dimension of the state.

Definition at line 110 of file PythonModelTemplate.py.

48.39.3.15  def PythonModelTemplate::PythonModelTemplate::GetParameter ( self, i )

Gets the i-th uncertain parameter.

Parameters
   in | i | parameter index.

Returns
   The vector associated with the i-th parameter.

Definition at line 181 of file PythonModelTemplate.py.

48.39.3.16  def PythonModelTemplate::PythonModelTemplate::GetParameterCorrelation ( self, i )

Returns the correlation between the uncertain parameters.

Parameters
   in | parameter index.
Returns

The correlation between the uncertain parameters.

Definition at line 201 of file PythonModelTemplate.py.

48.39.3.17  def PythonModelTemplate::PythonModelTemplate::GetParameterName ( self, i )

Returns the name of a parameter to be perturbed.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>parameter index</td>
</tr>
</tbody>
</table>

Returns

The name of the parameter.

Definition at line 194 of file PythonModelTemplate.py.

48.39.3.18  def PythonModelTemplate::PythonModelTemplate::GetParameterOption ( self, i )

Returns the perturbation option of the i-th parameter.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>parameter index</td>
</tr>
</tbody>
</table>

Returns

The perturbation option of the i-th parameter.

Definition at line 222 of file PythonModelTemplate.py.

48.39.3.19  def PythonModelTemplate::PythonModelTemplate::GetParameterPDF ( self, i )

Returns the PDF of the i-th parameter.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>parameter index</td>
</tr>
</tbody>
</table>

Returns

The PDF of the i-th parameter.

Definition at line 174 of file PythonModelTemplate.py.

48.39.3.20  def PythonModelTemplate::PythonModelTemplate::GetParameterPDFData ( self, i )

Returns parameters associated with the PDF of some model parameter.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>parameter index</td>
</tr>
</tbody>
</table>
48.39 PythonModelTemplate::PythonModelTemplate Class Reference 419

Returns
The parameters associated with the i-th parameter.

Definition at line 208 of file PythonModelTemplate.py.

48.39.3.21  def PythonModelTemplate::PythonModelTemplate::GetParameterVariance ( self, i )

Returns the covariance matrix associated with the i-th parameter.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>i</th>
<th>parameter index</th>
</tr>
</thead>
</table>

Returns
The covariance matrix associated with the i-th parameter.

Definition at line 215 of file PythonModelTemplate.py.

48.39.3.22  def PythonModelTemplate::PythonModelTemplate::GetState ( self )

Provides the controlled state vector.

Returns
The controlled state vector.

Definition at line 122 of file PythonModelTemplate.py.

48.39.3.23  def PythonModelTemplate::PythonModelTemplate::GetStateErrorVariance ( self )

Returns the state error variance.

Returns
The state error variance.

Definition at line 231 of file PythonModelTemplate.py.

48.39.3.24  def PythonModelTemplate::PythonModelTemplate::GetStateErrorVarianceInverse ( self )

Returns the inverse of the background error variance \(( B^{-1} )\).

Returns
The inverse of the background error variance \(( B^{-1} )\).

Definition at line 258 of file PythonModelTemplate.py.

48.39.3.25  def PythonModelTemplate::PythonModelTemplate::GetStateErrorVarianceProjector ( self )

Returns the matrix \( L \) in the decomposition of the state error covariance matrix \(( B)\) as a product \(LUL^T\).

Returns
The matrix \( L \).

Definition at line 245 of file PythonModelTemplate.py.
48.39.3.26 def PythonModelTemplate::PythonModelTemplate::GetStateErrorVarianceReduced (self)

Returns the matrix \( U \) in the decomposition of the state error covariance matrix \( B \) as a product \( LUL^T \).

Returns

The matrix \( U \).

Definition at line 252 of file PythonModelTemplate.py.

48.39.3.27 def PythonModelTemplate::PythonModelTemplate::GetStateErrorVarianceRow (self, row)

Returns a row of the state error variance.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>row</th>
<th>row index.</th>
</tr>
</thead>
</table>

Returns

The row with index \( row \) in the state error variance.

Definition at line 238 of file PythonModelTemplate.py.

48.39.3.28 def PythonModelTemplate::PythonModelTemplate::GetStateLowerBound (self)

Provides the state lower bound.

Returns

lower_bound the state lower bound (componentwise).

Definition at line 132 of file PythonModelTemplate.py.

48.39.3.29 def PythonModelTemplate::PythonModelTemplate::GetStateUpperBound (self, state_upper_bound)

Provides the state upper bound.

Returns

lower_bound the state upper bound (componentwise).

Definition at line 138 of file PythonModelTemplate.py.

48.39.3.30 def PythonModelTemplate::PythonModelTemplate::GetTangentLinearOperator (self)

Returns the tangent linear model.

Parameters

| out | \( M \) | the matrix of the tangent linear model. |

Definition at line 74 of file PythonModelTemplate.py.
Returns the current time.

Returns

The current time.

Definition at line 98 of file PythonModelTemplate.py.

Checks whether the model has finished.

Returns

True if the simulation is done, false otherwise.

Definition at line 80 of file PythonModelTemplate.py.

Initializes the model.

Parameters

|                | path          | The path to the configuration file. |

Definition at line 32 of file PythonModelTemplate.py.

Initializes the current time step for the model.

Definition at line 37 of file PythonModelTemplate.py.

Receives and handles a message.

Parameters

|                | message       | the received message |

Definition at line 269 of file PythonModelTemplate.py.

Triggers the actions after an uncertain parameter has been updated.

Parameters

|                | i             | parameter index. |

Definition at line 187 of file PythonModelTemplate.py.
def PythonModelTemplate::PythonModelTemplate::SetTime ( self, time )

Sets the current time.

Parameters

| in  | time | the current time. |

Definition at line 104 of file PythonModelTemplate.py.

def PythonModelTemplate::PythonModelTemplate::StateUpdated ( self )

Performs some calculations when the update of the model state is done.

Definition at line 126 of file PythonModelTemplate.py.

The documentation for this class was generated from the following file:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/model/PythonModelTemplate.py

## 48.40 Verdandi::PythonObservationManager Class Reference

This class is an interface for Python observation managers.

```cpp
#include <PythonObservationManager.hxx>
```

Inheritance diagram for Verdandi::PythonObservationManager:

```
Verdandi::VerdandiBase

Verdandi::PythonObservationManager
```

### Public Types

- typedef Matrix< double > tangent_linear_operator
  
  Type of the tangent linear operator.

- typedef Vector< double > tangent_linear_operator_row
  
  Type of a row of the tangent linear operator.

- typedef Matrix< double > error_variance
  
  Type of the observation error covariance matrix.

- typedef Vector< double > observation
  
  Type of the observation vector.

### Public Member Functions

- **PythonObservationManager ()**
  
  Constructor.

- **~PythonObservationManager ()**
  
  Destructor.

- template< class Model >
  
  void Initialize ( const Model &model, string configuration_file)

  Initializes the observation manager.
void DiscardObservation (bool discard_observation)

Activates or deactivates the option ‘discard_observation’.

– void SetTime (const Model &model, double time)

Sets the time of observations to be loaded.

– observation & GetObservation ()

Returns the observations.

– template<class state>
observation & GetInnovation (const state &x)

Returns an innovation.

– bool HasObservation () const

Indicates if some observations are available at current time.

– int GetNobservation () const

Returns the number of available observations.

– template<class state>
void ApplyOperator (const state &x, observation &y) const

Applies the observation operator to a given vector.

– template<class state>
void ApplyTangentLinearOperator (const state &x, observation &y) const

Applies the tangent linear operator to a given vector.

– double GetTangentLinearOperator (int i, int j) const

Returns an element of the tangent linear operator.

– tangent_linear_operator_row & GetTangentLinearOperatorRow (int row)

Returns a row of the tangent linear operator.

– const tangent_linear_operator & GetTangentLinearOperator ()

Returns the tangent linear operator.

– template<class state>
void ApplyAdjointOperator (const state &x, observation &y) const

Applies the adjoint operator to a given vector.

– double GetErrorVariance (int i, int j) const

Return an observation error covariance.

– const error_variance & GetErrorVariance ()

Returns the observation error variance.

– const error_variance & GetErrorVarianceInverse ()

Returns the inverse of the observation error covariance matrix.

– string GetName () const

Returns the name of the class.

– void Message (string message)

Receives and handles a message.

Static Public Member Functions

– static void StaticMessage (void *object, string message)

Receives and handles a message with a static method.
Protected Attributes

- PyObject * pyObservationManagerInstance_
  Instance of the Python model class.
- PyObject * pyObservationManagerFile_
  Name of the python model file.
- PyObject * pyObservationManagerModule_
  Imported model module from Python.
- PyObject * pyObservationManagerDict_
  Dictionary object that describes the module’s namespace.
- PyObject * pyObservationManagerClass_
  Model class from the module.
- bool is_module_initialized_
  Has the Python module been initialized?
- string module_
  Name of the Python module that contains the observation manager.
- string directory_
  Directory to include in *sys.path* for the module to be found. If no directory is to be added to *sys.path*, this attribute remains empty.
- string class_name_
  Name of the Python observation manager class.
- observation observation_
  Observation currently stored.
- observation innovation_
  Innovation currently stored.
- tangent_linear_operator tangent_operator_matrix_
  Tangent operator matrix (H).
- error_variance error_variance_
  Observation error covariance matrix (R).
- error_variance error_variance_inverse_
  Inverse of the observation error covariance matrix (R).
- int current_row_
  Index of the row of H currently stored.
- tangent_linear_operator_row tangent_operator_row_
  Value of the row of H currently stored.

48.40.1 Detailed Description

This class is an interface for Python observation managers.

48.40.2 Member Function Documentation

48.40.2.1 template< class state > void Verdandi::PythonObservationManager::ApplyAdjointOperator ( const state & x, observation & y ) const

Applies the adjoint operator to a given vector.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x</th>
<th>a vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>y</td>
<td>the value of the operator at x. It is resized if needed.</td>
</tr>
</tbody>
</table>
48.40.2.2 template<class state> void Verdandi::PythonObservationManager::ApplyOperator ( const state & x, observation & y ) const

Applies the observation operator to a given vector.
This method is called after 'SetTime' set the time at which the operator is defined.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x</th>
<th>a vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>y</td>
<td>the value of the operator applied to x. It is resized if needed.</td>
</tr>
</tbody>
</table>

Definition at line 310 of file PythonObservationManager.cxx.

48.40.2.3 template<class state> void Verdandi::PythonObservationManager::ApplyTangentLinearOperator ( const state & x, observation & y ) const

Applies the tangent linear operator to a given vector.
This method is called after 'SetTime' set the time at which the operator is defined.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x</th>
<th>a vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>y</td>
<td>the value of the tangent linear operator applied to x. It is resized if needed.</td>
</tr>
</tbody>
</table>

Definition at line 351 of file PythonObservationManager.cxx.

48.40.2.4 void Verdandi::PythonObservationManager::DiscardObservation ( bool discard_observation )

Activates or deactivates the option 'discard_observation'.

Parameters

| in | discard_observation | if set to true, each observation will be used at most one time. |

Definition at line 132 of file PythonObservationManager.cxx.

48.40.2.5 double Verdandi::PythonObservationManager::GetErrorVariance ( int i, int j ) const

Return an observation error covariance.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>i</th>
<th>row index.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>j</td>
<td>column index.</td>
</tr>
</tbody>
</table>

Returns

The element (i, j) of the observation error variance.

Definition at line 540 of file PythonObservationManager.cxx.
48.40.2.6 const PythonObservationManager::error_variance & Verdandi::PythonObservationManager::GetErrorVariance()

Returns the observation error variance.

Returns
  The observation error covariance matrix.

Definition at line 563 of file PythonObservationManager.cxx.

48.40.2.7 const PythonObservationManager::error_variance & Verdandi::PythonObservationManager::GetErrorVarianceInverse()

Returns the inverse of the observation error covariance matrix.

Returns
  The inverse of the matrix of the observation error covariance.

Definition at line 599 of file PythonObservationManager.cxx.

48.40.2.8 template< class state > PythonObservationManager::observation & Verdandi::PythonObservationManager::GetInnovation( const state & x )

Returns an innovation.
This method is called after ‘SetTime’ set the time at which the innovation is requested.

Parameters

| in | state | state vector. |

Returns
  The innovation vector.

Definition at line 218 of file PythonObservationManager.cxx.

48.40.2.9 string Verdandi::PythonObservationManager::GetName( ) const [virtual]

Returns the name of the class.

Returns
  The name of the class.

Reimplemented from Verdandi::VerdandiBase.

Definition at line 635 of file PythonObservationManager.cxx.

48.40.2.10 int Verdandi::PythonObservationManager::GetNobservation( ) const

Returns the number of available observations.

Returns
  The total number of observation at current time.

Definition at line 280 of file PythonObservationManager.cxx.
48.40.2.11 PythonObservationManager::observation & Verdandi::PythonObservationManager::GetObservation ( )

Returns the observations.
This method is called after ‘SetTime’ set the time at which the observations are requested.

Returns
The observation vector.

Definition at line 177 of file PythonObservationManager.cxx.

48.40.2.12 double Verdandi::PythonObservationManager::GetTangentLinearOperator ( int i, int j ) const

Returns an element of the tangent linear operator.
This method is called after ‘SetTime’ set the time at which the operator is defined.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>i</th>
<th>row index.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>j</td>
<td>column index.</td>
</tr>
</tbody>
</table>

Returns
The element \((i, j)\) of the tangent linear operator.

Definition at line 391 of file PythonObservationManager.cxx.

48.40.2.13 const PythonObservationManager::tangent_linear_operator& Verdandi::PythonObservationManager::-
GetTangentLinearOperator ( )

Returns the tangent linear operator.
This method is called after ‘SetTime’ set the time at which the operator is defined.

Returns
The matrix of the tangent linear operator.

Definition at line 461 of file PythonObservationManager.cxx.

48.40.2.14 PythonObservationManager::tangent_linear_operator_row & Verdandi::PythonObservationManager::-
GetTangentLinearOperatorRow ( int row )

Returns a row of the tangent linear operator.
This method is called after ‘SetTime’ set the time at which the operator is defined.

Parameters

| in | row | row index. |

Returns
The row \(row\) of the tangent linear operator.

Definition at line 417 of file PythonObservationManager.cxx.
48.40.2.15 template < class Model > void Verdi::PythonObservationManager::Initialize ( const Model & model, string configuration_file )

Initializes the observation manager.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>model</th>
<th>model.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>configuration_file</td>
<td>configuration file.</td>
</tr>
</tbody>
</table>

Template Parameters

- Model the model type; e.g. ShallowWater<double>

Definition at line 73 of file PythonObservationManager.cxx.

48.40.2.16 template < class Model > void Verdi::PythonObservationManager::SetTime ( const Model & model, double time )

Sets the time of observations to be loaded.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>model</th>
<th>the model.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>time</td>
<td>a given time.</td>
</tr>
</tbody>
</table>

Definition at line 154 of file PythonObservationManager.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdi/verdandi-1.7/observation_manager/PythonObservationManager.hxx
- /nas/home2/n/nclaude/Travail/code/Verdi/verdandi-1.7/observation_manager/PythonObservationManager.cxx

48.41 PythonObservationManagerTemplate::PythonObservationManagerTemplate Class Reference

This class is a template observation manager in Python.

Public Member Functions

- def __init__
  
  Initializes the observation manager.
- def DiscardObservation
  
  Activates or deactivates the option 'discard_observation'.
- def SetTime
  
  Sets the time of observations to be loaded.
- def GetObservation
  
  Returns the observations.
- def GetInnovation
  
  Returns an innovation This method is called after 'SetTime' set the time at which the innovation is requested.
- def HasObservation
  
  Indicates if some observations are available at a given time or at current time.
def GetNobservation
Returns the number of available observations.

def ApplyOperator
Applies the observation operator to a given vector.

def ApplyTangentLinearOperator
Applies the tangent linear operator to a given vector.

def GetTangentLinearOperator
Returns an element of the tangent linear operator, or the full tangent linear operator if no row and column index is provided.

def GetErrorVariance
Returns an element of the observation error covariance matrix, or the full matrix.

def GetErrorVarianceInverse
Returns the inverse of the observation error covariance matrix.

def GetName
Returns the name of the class.

def Message
Receives and handles a message.

48.41.1 Detailed Description
This class is a template observation manager in Python.

48.41.2 Constructor & Destructor Documentation
48.41.2.1 def PythonObservationManagerTemplate::PythonObservationManagerTemplate::init(self, path, Nstate_model)
Initializes the observation manager.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>path</th>
<th>The path to the python file</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>Nstate_model</td>
<td>The size of the state vector model</td>
</tr>
</tbody>
</table>

Definition at line 28 of file PythonObservationManagerTemplate.py.

48.41.3 Member Function Documentation
48.41.3.1 def PythonObservationManagerTemplate::PythonObservationManagerTemplate::ApplyOperator(self, state, y = None)
Applies the observation operator to a given vector.
This method is called after ‘SetTime’ set the time at which the operator is defined.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>x</th>
<th>a vector.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>y</td>
<td>the value of the operator applied to x.</td>
</tr>
</tbody>
</table>

Definition at line 92 of file PythonObservationManagerTemplate.py.
48.41.3.2  def PythonObservationManagerTemplate::PythonObservationManagerTemplate::ApplyTangentLinearOperator ( self, x )

Applies the tangent linear operator to a given vector.
This method is called after 'SetTime' set the time at which the operator is defined.

Parameters

in  x  a vector to which the tangent linear operator should be applied

Returns

The value of the operator applied to x.

Definition at line 102 of file PythonObservationManagerTemplate.py.

48.41.3.3  def PythonObservationManagerTemplate::PythonObservationManagerTemplate::DiscardObservation ( discard_observation )

Activates or deactivates the option 'discard_observation'.

Parameters

in  discard_ - observation  if set to true, each observation will be used at most one time.

Definition at line 35 of file PythonObservationManagerTemplate.py.

48.41.3.4  def PythonObservationManagerTemplate::PythonObservationManagerTemplate::GetErrorVariance ( self, i = None, j = None )

Returns an element of the observation error covariance matrix, or the full matrix.

Parameters

in  i  row index.
in  j  column index.

Returns

The element (i, j) of the observation error covariance matrix, or the full matrix.

Definition at line 144 of file PythonObservationManagerTemplate.py.

48.41.3.5  def PythonObservationManagerTemplate::PythonObservationManagerTemplate::GetErrorVarianceInverse ( self )

Returns the inverse of the observation error covariance matrix.

Returns

The inverse of the observation error covariance matrix.

Definition at line 150 of file PythonObservationManagerTemplate.py.

48.41.3.6  def PythonObservationManagerTemplate::PythonObservationManagerTemplate::GetInnovation ( self, state )

Returns an innovation This method is called after 'SetTime' set the time at which the innovation is requested.
Parameters

| in | state | state vector. |

Returns

innovation vector.

Definition at line 64 of file PythonObservationManagerTemplate.py.

```python
48.41.3.7 def PythonObservationManagerTemplate::PythonObservationManagerTemplate::GetName ( self )

Returns the name of the class.

Returns

The name of the class.

Definition at line 156 of file PythonObservationManagerTemplate.py.

```python
48.41.3.8 def PythonObservationManagerTemplate::PythonObservationManagerTemplate::GetNobservation ( self )

Returns the number of available observations.

Returns

The total number of observation at current time.

Definition at line 80 of file PythonObservationManagerTemplate.py.

```python
48.41.3.9 def PythonObservationManagerTemplate::PythonObservationManagerTemplate::GetObservation ( self )

Returns the observations.

This method is called after 'SetTime' set the time at which the observations are requested.

Returns

observation observation vector.

Definition at line 52 of file PythonObservationManagerTemplate.py.

```python
48.41.3.10 def PythonObservationManagerTemplate::PythonObservationManagerTemplate::GetTangentLinearOperator
            ( self, i = None, j = None, return, def, GetTangentLinearOperatorRow, self, row, return, def,
                ApplyAdjointOperator, self, x )

Returns an element of the tangent linear operator, or the full tangent linear operator if no row and column index is provided.

This method is called after 'SetTime' set the time at which the operator is defined.

Parameters

| in | i | row index. |
| in | j | column index. |
Returns

The element \((i, j)\) of the tangent linear operator.

Definition at line 113 of file PythonObservationManagerTemplate.py.

### 48.41.3.11 def PythonObservationManagerTemplate::PythonObservationManagerTemplate::HasObservation ( self, time = None )

Indicates if some observations are available at a given time or at current time.

#### Parameters

| in    | time | a given time. |

Definition at line 74 of file PythonObservationManagerTemplate.py.

### 48.41.3.12 def PythonObservationManagerTemplate::PythonObservationManagerTemplate::Message ( self, message )

Receives and handles a message.

#### Parameters

| in     | message | the received message |

Definition at line 162 of file PythonObservationManagerTemplate.py.

### 48.41.3.13 def PythonObservationManagerTemplate::PythonObservationManagerTemplate::SetTime ( self, time )

Sets the time of observations to be loaded.

#### Parameters

| in    | time | The current time. |

Definition at line 41 of file PythonObservationManagerTemplate.py.

The documentation for this class was generated from the following file:

– /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/observation_manager/PythonObservationManagerTemplate.py

### 48.42 Verdandi::QuadraticModel Class Reference

This class is a quadratic model.

```cpp
#include <QuadraticModel.hxx>
```

Inheritance diagram for Verdandi::QuadraticModel:

```
Verdandi::VerdandiBase

```

```none
Verdandi::QuadraticModel
```
Public Types

- typedef T value_type
- typedef T * pointer
- typedef const T * const_pointer
- typedef T & reference
- typedef const T & const_reference
- typedef Matrix<T> tangent_linear_operator
- typedef Matrix<T> state_error_variance
- typedef Matrix<T> state_error_variance_reduced
- typedef Vector<T> state_error_variance_row
- typedef Vector<T> state
- typedef Matrix<T> matrix_state_observation
- typedef Matrix<T> error_variance
- typedef Matrix<T, Symmetric, Seldon::RowSymPacked> parameter_variance

Public Member Functions

- QuadraticModel ()
  Constructor.
- QuadraticModel (string configuration_file)
  Constructor.
- ~QuadraticModel ()
  Destructor.
- void Initialize (string configuration_file)
  Initializes the model.
- void InitializeStep ()
  Initializes the current time step for the model.
- void Forward ()
  Advances one step forward in time.
- double ApplyOperator (state &x, bool preserve_state=true)
  Applies the model to a given state vector.
- double ApplyTangentLinearOperator (state &x)
  Applies the tangent linear model to a given vector.
- tangent_linear_operator& GetTangentLinearOperator ()
  Returns the tangent linear model.
- bool HasFinished () const
  Checks whether the model has finished.
- void Save ()
  Saves the simulated data.
- void FinalizeStep ()
  Finalizes the current time step for the model.
- void Finalize ()
  Finalizes the model.
- T GetDelta_t () const
  Returns the time step.
- double GetTime () const
  Returns the current time.
- void SetTime (double time)
  Sets the current time.
- int GetNstate () const
Returns the dimension of the state.

- **int GetNfull_state () const**
  Returns the dimension of the full state.

- **state & GetState ()**
  Provides the controlled state vector.

- **void StateUpdated ()**
  Performs some calculations when the update of the model state is done.

- **state & GetFullState ()**
  Provides the full state vector.

- **void FullStateUpdated ()**
  Performs some calculations when the update of the full model state is done.

- **void SetSource (state &source)**
  Allows to set a source to the model.

- **int GetNparameter ()**
  Returns the number of parameters to be perturbed.

- **uncertain_parameter & GetParameter (int i)**
  Gets the i-th uncertain parameter.

- **void ParameterUpdated (int i)**

- **string GetParameterName (int i)**
  Returns the name of a parameter to be perturbed.

- **Vector<T> & GetParameterCorrelation (int i)**
  Returns the correlation between the uncertain parameters.

- **string GetParameterPDF (int i)**
  Returns the PDF of the i-th parameter.

- **parameter_variance & GetParameterVariance (int i)**
  Returns the covariance matrix associated with the i-th parameter.

- **Vector<T> & GetParameterPDFData (int i)**
  Returns parameters associated with the PDF of some model parameter.

- **string GetParameterOption (int i)**
  Returns the perturbation option of the i-th parameter.

- **error_variance & GetErrorVariance ()**
  Returns the model error variance.

- **const error_variance & GetErrorVariance () const**
  Returns the model error variance.

- **error_variance & GetErrorVarianceSqrt ()**
  Returns the square root of the model error variance.

- **const error_variance & GetErrorVarianceSqrt () const**
  Returns the square root of the model error variance.

- **state_error_variance & GetStateErrorVariance ()**
  Returns the state error variance.

- **const state_error_variance & GetStateErrorVariance () const**
  Returns the state error variance.

- **state_error_variance_row & GetStateErrorVarianceRow (int row)**
  Returns a row of the state error variance.

- **state_error_variance & GetStateErrorVarianceSqrt ()**
  Returns the square root of the state error variance.

- **const state_error_variance & GetStateErrorVarianceSqrt () const**
  Returns the square root of the state error variance.

- **state_error_variance & GetStateErrorVarianceProjector ()**
  Returns the state error variance projector.

- **state_error_variance_reduced & GetStateErrorVarianceReduced ()**
  Returns the reduced error state variance.
string GetName () const

Returns the name of the class.

void Message (string message)

Receives and handles a message.

Static Public Member Functions

static void StaticMessage (void *object, string message)

Receives and handles a message with a static method.

Protected Attributes

int Nstate_

Dimension of the state.

Vector<T> state_

State vector.

state source_

Source vector.

bool with_quadratic_term_

Should the quadratic term be applied?

bool with_linear_term_

Should the linear term be applied?

bool with_constant_term_

Should the constant term be applied?

vector<Matrix<T>> S_

Quadratic terms.

Matrix<T> L_

Matrix that defines the linear part of the model.

Vector<T> b_

Vector that defines the constant part of the model.

double Delta_t_

Time step.

double final_time_

Final time of the simulation.

double time_

Current time.

Vector<T> S_state_

Temporary variable that stores S times the state vector.

bool is_quadratic_perturbed_

Should the quadratic term be perturbed?

bool is_linear_perturbed_

Should the linear term be perturbed?

bool is_constant_perturbed_

Should the constant term be perturbed?

vector<uncertain_parameter *> parameter_

Parameters to be perturbed.

vector<string> uncertain_parameter_vector_

List of parameters to be perturbed.

int Nparameter_

Number of parameters to be perturbed.
- `vector<` string > `parameter_name_`
  Name of the parameters to be perturbed.
- `vector<` Vector< T > > `correlation_`
  Correlations between the constant term and the other terms.
- `vector<` string > `pdf_`
  Name of the probability distribution for the constant term.
- `vector<` Vector< T > > `mean_`
  Mean of the probability distribution for the constant term.
- `vector<` Matrix< T, Symmetric, RowSymPacked > > `variance_`
  Covariance matrix for the constant term.
- `vector<` Vector< T > > `optional_parameters_`
  PDF parameters for the constant term.
- `error_variance Q_`
  Variance of the model error.
- `error_variance Q_sqrt_`
  Variance of the model error in square root form.
- `state_error_variance P_`
  Variance of the state error.
- `state_error_variance state_error_variance_projector_`
  Projector matrix \( L \) in the decomposition of the background error covariance matrix \( B \) as a product \( LUL^T \).
- `state_error_variance_reduced state_error_variance_reduced_`
  Reduced matrix \( U \) in the decomposition of the background error covariance matrix \( B \) as a product \( LUL^T \).
- `state_error_variance P_sqrt_`
  Variance of the state error in square root form.
- `tangent_linear_operator tangent_linear_operator_`
  Tangent linear operator \( (H) \).
- `int current_row_`
  Index of the row of \( P \) currently stored.
- `state_error_variance_row state_error_variance_row_`
  Value of the row of \( P \) currently stored.
- `OutputSaver output_saver_`
  Output saver.

### 48.42.1 Detailed Description

This class is a quadratic model. The model is defined as
\[
\frac{dx}{dt} = \begin{bmatrix} Q_i & L_i \\ \end{bmatrix} x + b, \]
where \( Q_i \) is a matrix, \( L_i \) is the \( i \)-th row of the matrix \( L \) and \( b \) a vector.

**Template Parameters**

\[
T \quad \text{the type of floating-point numbers.}
\]

### 48.42.2 Constructor & Destructor Documentation

#### 48.42.2.1 Verdandi::QuadraticModel::QuadraticModel ( string configuration_file )

Constructor.
It reads the initial condition and the time settings.
Parameters

|      | configuration_file | path to the configuration file. |

Definition at line 54 of file QuadraticModel.cxx.

### 48.42.3 Member Function Documentation

#### 48.42.3.1 double Verdandi::QuadraticModel::ApplyOperator (state & x, bool preserve_state = true )

Applies the model to a given state vector.

Parameters

| in, out | x | on entry, the state vector to which the model is applied; on exit, the state vector after the model is applied. |
| in       | forward | Boolean to indicate if the model has to go on to the next step. |
| in       | preserve_state | Boolean to indicate if the model state has to be preserved. |

Returns

The time associated with x on exit plus one time step.

Warning

The time of the model has to be preserved.

Definition at line 519 of file QuadraticModel.cxx.

#### 48.42.3.2 double Verdandi::QuadraticModel::ApplyTangentLinearOperator (state & x )

Applies the tangent linear model to a given vector.

Parameters

| in, out | x | on entry, a vector to which the tangent linear model should be applied; on exit, the result. |

Returns

The time associated with x on exit. This time should be the model time plus one time step.

Definition at line 550 of file QuadraticModel.cxx.

#### 48.42.3.3 T Verdandi::QuadraticModel::GetDelta_t () const

Returns the time step.

Returns

The time step.

Definition at line 653 of file QuadraticModel.cxx.

#### 48.42.3.4 QuadraticModel<T>::error_variance & Verdandi::QuadraticModel::GetErrorVariance ( )

Returns the model error variance.
Returns

The model error variance.

Note

If the matrix is empty, it is then assumed there is no model error.

Definition at line 897 of file QuadraticModel.cxx.

48.42.3.5  const QuadraticModel<& T>::error_variance & Verdandi::QuadraticModel::GetErrorVariance ( ) const

Returns the model error variance.

Returns

The model error variance.

Note

If the matrix is empty, it is then assumed there is no model error.

Definition at line 911 of file QuadraticModel.cxx.

48.42.3.6  QuadraticModel<& T>::error_variance & Verdandi::QuadraticModel::GetErrorVarianceSqrt ( )

Returns the square root of the model error variance.

Returns

The square root of the model error variance.

Definition at line 923 of file QuadraticModel.cxx.

48.42.3.7  const QuadraticModel<& T>::error_variance & Verdandi::QuadraticModel::GetErrorVarianceSqrt ( ) const

Returns the square root of the model error variance.

Returns

The square root of the model error variance.

Definition at line 935 of file QuadraticModel.cxx.

48.42.3.8  QuadraticModel<& T>::state & Verdandi::QuadraticModel::GetFullState ( )

Provides the full state vector.

Returns

state the full state vector.

Definition at line 728 of file QuadraticModel.cxx.
string QuadraticModel::GetName() const [virtual]

Returns the name of the class.

Returns
The name of the class.

Reimplemented from Verdandi::VerdandiBase.
Definition at line 1024 of file QuadraticModel.cxx.

int QuadraticModel::GetNfull_state() const

Returns the dimension of the full state.

Returns
The dimension of the full state.

Definition at line 697 of file QuadraticModel.cxx.

int QuadraticModel::GetNparameter()

Returns the number of parameters to be perturbed.

Returns
The number of parameters to be perturbed.

Definition at line 757 of file QuadraticModel.cxx.

int QuadraticModel::GetNstate() const

Returns the dimension of the state.

Returns
The dimension of the state.

Definition at line 686 of file QuadraticModel.cxx.

QuadraticModel< T >::uncertain_parameter & QuadraticModel::GetParameter(int i)

Gets the i-th uncertain parameter.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>i</th>
<th>index of the parameter.</th>
</tr>
</thead>
</table>

Returns
The vector associated with the i-th parameter.

Definition at line 770 of file QuadraticModel.cxx.
Returns the correlation between the uncertain parameters.
Since there is only one parameter, an empty vector is returned.

Parameters

| in  | i | parameter index |

Returns

An empty vector.

Definition at line 807 of file QuadraticModel.hxx.

48.42.3.15 string Verdandi::QuadraticModel::GetParameterName ( int i )

Returns the name of a parameter to be perturbed.

Parameters

| in  | i | index of the parameter |

Returns

The name of the parameter.

Definition at line 794 of file QuadraticModel.hxx.

48.42.3.16 string Verdandi::QuadraticModel::GetParameterOption ( int i )

Returns the perturbation option of the i-th parameter.

Parameters

| in  | i | parameter index |

Returns

The perturbation option of the i-th parameter.

Definition at line 858 of file QuadraticModel.hxx.

48.42.3.17 string Verdandi::QuadraticModel::GetParameterPDF ( int i )

Returns the PDF of the i-th parameter.

Parameters

| in  | i | uncertain-variable index |

Returns

The PDF of the i-th parameter.

Definition at line 819 of file QuadraticModel.hxx.
48.42.3.18  Vector< T > & Verdandi::QuadraticModel::GetParameterPDFData ( int i )

Returns parameters associated with the PDF of some model parameter.
In case of normal or log-normal distribution, the parameters are clipping parameters.

Parameters

| in | i | model parameter index. |

Returns

The parameters associated with the i-th parameter.

Definition at line 846 of file QuadraticModel.cxx.

48.42.3.19  QuadraticModel< T >::parameter_variance & Verdandi::QuadraticModel::GetParameterVariance ( int i )

Returns the covariance matrix associated with the i-th parameter.

Parameters

| in | i | parameter index. |

Returns

The covariance matrix associated with the i-th parameter.

Definition at line 833 of file QuadraticModel.cxx.

48.42.3.20  QuadraticModel< T >::state & Verdandi::QuadraticModel::GetState ( )

Provides the controlled state vector.

Returns

state the controlled state vector.

Definition at line 709 of file QuadraticModel.cxx.

48.42.3.21  QuadraticModel< T >::state_error_variance & Verdandi::QuadraticModel::GetStateErrorVariance ( )

Returns the state error variance.

Returns

The state error variance.

Definition at line 947 of file QuadraticModel.cxx.

48.42.3.22  const QuadraticModel< T >::state_error_variance & Verdandi::QuadraticModel::GetStateErrorVariance ( ) const

Returns the state error variance.

Returns

The state error variance.

Definition at line 959 of file QuadraticModel.cxx.
48.42.3.23 QuadraticModel<T>::state_error_variance & Verdandi::QuadraticModel::GetStateErrorVarianceProjector()

Returns the state error variance projector.

Definition at line 1013 of file QuadraticModel.cxx.

48.42.3.24 QuadraticModel<T>::state_error_variance_reduced & Verdandi::QuadraticModel::GetStateErrorVarianceReduced()

Returns the reduced error state variance.

Definition at line 1001 of file QuadraticModel.cxx.

48.42.3.25 QuadraticModel<T>::state_error_variance_row & Verdandi::QuadraticModel::GetStateErrorVarianceRow(int row)

Returns a row of the state error variance.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>row</th>
<th>row index.</th>
</tr>
</thead>
</table>

Returns

The row with index row in the state error variance.

Definition at line 972 of file QuadraticModel.cxx.

48.42.3.26 QuadraticModel<T>::state_error_variance & Verdandi::QuadraticModel::GetStateErrorVarianceSqrt()

Returns the square root of the state error variance.

Definition at line 989 of file QuadraticModel.cxx.

48.42.3.27 QuadraticModel<T>::tangent_linear_operator & Verdandi::QuadraticModel::GetTangentLinearOperator()

Returns the tangent linear model.

Definition at line 577 of file QuadraticModel.cxx.
48.42.3.28  double Verdandi::QuadraticModel::GetTime ( ) const

Returns the current time.

Returns
    The current time.

Definition at line 664 of file QuadraticModel.cxx.

48.42.3.29  bool Verdandi::QuadraticModel::HasFinished ( ) const

Checks whether the model has finished.

Returns
    True if no more data assimilation is required, false otherwise.

Definition at line 613 of file QuadraticModel.cxx.

48.42.3.30  void Verdandi::QuadraticModel::Initialize ( string configuration_file )

Initializes the model.
It reads the initial condition and the time settings.

Parameters
    in  configuration_file  configuration file.

Definition at line 90 of file QuadraticModel.cxx.

48.42.3.31  void Verdandi::QuadraticModel::ParameterUpdated ( int i )

Performs some calculations when the update of the i-th uncertain parameter is done.

Parameters
    in  i  index of the parameter.

Definition at line 783 of file QuadraticModel.cxx.

48.42.3.32  void Verdandi::QuadraticModel::Save ( )

Saves the simulated data.
It saves the state.
Definition at line 623 of file QuadraticModel.cxx.

48.42.3.33  void Verdandi::QuadraticModel::SetSource ( state & source )

Allows to set a source to the model.

Parameters
    in  source  The source to be added.
48.42.3.34 void Verdandi::QuadraticModel::SetTime ( double time )

Sets the current time.

Parameters

| in   | time | the current time. |

Definition at line 675 of file QuadraticModel.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/model/QuadraticModel.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/model/QuadraticModel.cxx

48.43 QuadraticModel::QuadraticModel Class Reference

This class is a quadratic model written in Python.

Public Member Functions

- def __init__
  Constructs the model.
- def Initialize
  Initializes the model.
- def InitializeStep
  Initializes the current time step for the model.
- def Forward
  Processing ###.
- def ApplyOperator
  Applies the model to a given vector.
- def ApplyTangentLinearOperator
  Applies the tangent linear model to a given vector.
- def GetTangentLinearOperator
  Returns the tangent linear model.
- def HasFinished
  Checks whether the model has finished.
- def Save
  Saves the simulated data.
- def FinalizeStep
  Finalizes the current time step for the model.
- def Finalize
  Finalizes the model.
- def GetDelta_t
  Access methods ###.
- def GetTime
  Returns the current time.
- def SetTime
  Sets the current time.
- def GetNstate
  Returns the dimension of the state.
- def GetNFullstate
  Returns the dimension of the full state.
- def GetState
  Provides the controlled state vector.
- def StateUpdated
  Performs some calculations when the update of the model state is done.
- def GetFullState
  Provides the full state vector.
- def FullStateUpdated
  Performs some calculations when the update of the model state is done.
- def GetErrorVariance
  Errors ###.
- def GetErrorVarianceSqrt
  Returns the square root of the model error variance.
- def GetStateErrorVariance
  Returns the state error variance.
- def GetStateErrorVarianceRow
  Returns a row of the state error variance.
- def GetName
  Returns the name of the class.
- def Message
  Receives and handles a message.

Public Attributes

- state_
- Nstate_
- S_
- L_
- b_
- Delta_t_
- time_
- final_time_
- Q_
- Q_sqrt
- P_
- P_sqrt

48.43.1 Detailed Description

This class is a quadratic model written in Python.

48.43.2 Constructor & Destructor Documentation

48.43.2.1 def QuadraticModel::QuadraticModel:::__init__ ( self )

Constructs the model.

Definition at line 28 of file QuadraticModel.py.
Applies the model to a given vector. The current state of the model is modified.

Parameters

<table>
<thead>
<tr>
<th>in, out</th>
<th>state</th>
<th>on entry, the state vector to which the model is applied; on exit, the state vector after the model is applied.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>preserve_state</td>
<td>Boolean to indicate if the model state has to be preserved.</td>
</tr>
</tbody>
</table>

Returns

The time associated with state on exit plus one time step.

Definition at line 141 of file QuadraticModel.py.

Applies the tangent linear model to a given vector.

Parameters

| in, out | x | on entry, a vector to which the tangent linear model should be applied; on exit, the result. |

Returns

The time associated with x on exit. This time should be the model time plus one time step.

Definition at line 159 of file QuadraticModel.py.

Finalizes the model.

Definition at line 216 of file QuadraticModel.py.

Finalizes the current time step for the model.

Definition at line 212 of file QuadraticModel.py.

Advances one step forward in time.

\[ x_{h+1} = M_h(x_h, p_h) \]

Definition at line 115 of file QuadraticModel.py.
48.43.3.6  def QuadraticModel::QuadraticModel::FullStateUpdated ( self )

Performs some calculations when the update of the model state is done.
Definition at line 271 of file QuadraticModel.py.

48.43.3.7  def QuadraticModel::QuadraticModel::GetDelta_t ( self )

Access methods ###.
Returns the time step.
Returns
   The time step.
Definition at line 225 of file QuadraticModel.py.

48.43.3.8  def QuadraticModel::QuadraticModel::GetErrorVariance ( self )

Errors ###.
Returns the model error variance.
Returns
   The model error variance.
Definition at line 280 of file QuadraticModel.py.

48.43.3.9  def QuadraticModel::QuadraticModel::GetErrorVarianceSqrt ( self )

Returns the square root of the model error variance.
Returns
   The square root of the model error variance.
Definition at line 286 of file QuadraticModel.py.

48.43.3.10 def QuadraticModel::QuadraticModel::GetFullState ( self )

Provides the full state vector.
Returns
   The full state vector.
Definition at line 266 of file QuadraticModel.py.

48.43.3.11 def QuadraticModel::QuadraticModel::GetName ( self )

Returns the name of the class.
Returns
   The name of the class.
Definition at line 305 of file QuadraticModel.py.
48.43.3.12 def QuadraticModel::QuadraticModel::GetNFullstate ( self )

Returns the dimension of the full state.

Returns
  The dimension of the full state.

Definition at line 249 of file QuadraticModel.py.

48.43.3.13 def QuadraticModel::QuadraticModel::GetNstate ( self )

Returns the dimension of the state.

Returns
  The dimension of the state.

Definition at line 243 of file QuadraticModel.py.

48.43.3.14 def QuadraticModel::QuadraticModel::GetState ( self )

Provides the controlled state vector.

Returns
  The controlled state vector.

Definition at line 255 of file QuadraticModel.py.

48.43.3.15 def QuadraticModel::QuadraticModel::GetStateErrorVariance ( self )

Returns the state error variance.

Returns
  The state error variance.

Definition at line 292 of file QuadraticModel.py.

48.43.3.16 def QuadraticModel::QuadraticModel::GetStateErrorVarianceRow ( self, row )

Returns a row of the state error variance.

Parameters

| In   | row | row index. |

Returns
  The row with index row in the state error variance.

Definition at line 299 of file QuadraticModel.py.
48.43.3.17    def QuadraticModel::QuadraticModel::GetTangentLinearOperator ( self )

Returns the tangent linear model.

Parameters

| out  | $M$ | the matrix of the tangent linear model. |

Definition at line 179 of file QuadraticModel.py.

48.43.3.18    def QuadraticModel::QuadraticModel::GetTime ( self )

Returns the current time.

Returns

  The current time.

Definition at line 231 of file QuadraticModel.py.

48.43.3.19    def QuadraticModel::QuadraticModel::HasFinished ( self )

Checks whether the model has finished.

Returns

  True if the simulation is done, false otherwise.

Definition at line 203 of file QuadraticModel.py.

48.43.3.20    def QuadraticModel::QuadraticModel::Initialize ( self, path )

Initializes the model.

Parameters

| in     | $path$ | The path to the configuration file. |

Definition at line 34 of file QuadraticModel.py.

48.43.3.21    def QuadraticModel::QuadraticModel::InitializeStep ( self )

Initializes the current time step for the model.

Definition at line 106 of file QuadraticModel.py.

48.43.3.22    def QuadraticModel::QuadraticModel::Message ( self, message )

Receives and handles a message.

Parameters

| in    | $message$ | the received message |

Definition at line 311 of file QuadraticModel.py.
48.43.3.23  def QuadraticModel::QuadraticModel::Save ( self )

Saves the simulated data.
Definition at line 208 of file QuadraticModel.py.

48.43.3.24  def QuadraticModel::QuadraticModel::SetTime ( self, time )

Sets the current time.
Parameters

| in   | time | the current time. |

Definition at line 237 of file QuadraticModel.py.

48.43.3.25  def QuadraticModel::QuadraticModel::StateUpdated ( self )

Performs some calculations when the update of the model state is done.
Definition at line 260 of file QuadraticModel.py.

The documentation for this class was generated from the following file:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/model/QuadraticModel.py

48.44  Verdandi::ReducedMinimax Class Reference

This class implements a reduced minimax filter.
#include <ReducedMinimax.hxx>

Inheritance diagram for Verdandi::ReducedMinimax:

[Diagram showing inheritance structure]

Public Types

- typedef Model::state model_state
  Type of the model state vector.
- typedef Model::state_error_variance model_state_error_variance
  Type of the background error variance.
- typedef Model::error_variance model_error_variance
  Type of the background error variance.
- typedef ObservationManager::tangent_linear_operator observation_tangent_linear_operator
  Type of the observation operator.

Public Member Functions

- ReducedMinimax ()
Main constructor.
- `~ReducedMinimax()`  
Destructor.
- `void Initialize(string configuration_file, bool initialize_model=true, bool initialize_observation_manager=true)`  
Initializes the driver.
- `void Initialize(VerdandiOps &configuration, bool initialize_model=true, bool initialize_observation_manager=true)`  
Initializes the driver.
- `void InitializeStep()`  
Initializes a step.
- `void Forward()`  
Performs a step forward.
- `void FinalizeStep()`  
Finalizes a step for the model.
- `void Finalize()`  
- `void FilterInitialization()`  
Initialization of the filter.
- `void Propagation()`  
Propagates the state and the minimax gain.
- `void ComputeTangentLinearModel(Matrix<T, General, RowMajor> &M)`  
Computes the full matrix of the tangent linear model.
- `void SnapshotRecording()`  
Performs a step forward with snapshot recording.
- `bool HasFinished()`  
Checks whether the model has finished.
- `int GetMode() const`  
Returns the current mode (snapshot recording or minimax calculations).
- `Matrix<T, General, RowMajor> &GetProjection()`  
Returns the current projection matrix.
- `Matrix<T, General, RowMajor> &GetPreviousProjection()`  
Returns the previous projection matrix.
- `Model & GetModel()`  
Returns the model.
- `ObservationManager & GetObservationManager()`  
Returns the observation manager.
- `OutputSaver & GetOutputSaver()`  
Returns the output saver.
- `string GetName() const`  
Returns the name of the class.
- `void Message(string message)`  
Receives and handles a message.

Static Public Member Functions

- `static void StaticMessage(void *object, string message)`  
Receives and handles a message with a static method.
Protected Attributes

- Model model_
  Underlying model.
- ObservationManager observation_manager_
  Observation manager.
- string configuration_file_
  Path to the configuration file.
- bool display_iteration_
  Should the iterations be displayed?
- bool display_time_
  Should the current time be displayed?
- bool with_filtering_
  Should the filter be applied?
- string model_configuration_file_
  Model configuration file.
- string observation_configuration_file_
  Configuration file for the observation manager.
- int Nstate_
  Dimension of the state.
- int Nobservation_
  Number of observations.
- int Nstep_snapshot_
  Width of the simulation window (in number of time steps) for the snapshot recording.
- bool with_Hty_HtHx_
  Should $H^T y$ and $H^T H x$ be included in the snapshots?
- double Nobservation_step_per_snapshot_
  Approximate number of observations per state snapshots.
- Matrix< T, General, RowMajor > snapshot_
  Snapshots to be used for POD.
- int Nsnapshot_
  Current number of snapshots stored in matrix 'snapshot_'.
- Vector< T > singular_value_
  Singular values.
- Matrix< T, General, RowMajor > left_singular_vector_
  Left singular vectors.
- Matrix< T, General, RowMajor > right_singular_vector_
  Right singular vectors.
- string reduction_method_
  Reduction method: "none" or "pod".
- int Nprojection_max_
  Maximum dimension of the reduced space.
- T acceptable_error_
  Acceptable relative quadratic error.
- int Nprojection_
  Dimension of the reduced space.
- Matrix< T, General, RowMajor > projection_
  Projection matrix.
- int Nprevious_projection_
  Previous dimension of the reduced space.
- Matrix< T, General, RowMajor > previous_projection_
Previous projection matrix.

- **int iteration_**
  Global iteration.

- **int mode_**
  Mode: 0 for snapshot recording for POD, 1 for error computation.

- **int inner_iteration_**
  Number of iteration in a given sequence. After 'nsnapshot_' iterations, one switches to either snapshot recording or error computation.

- **bool first_sequence_**
  Is the current sequence in the first window?

- **double starting_time_**
  Starting time of a new sequence of error computation.

- **model_state full_state_**
  Snapshot of the full state vector used to restart before an error computation sequence. This full state vector is a snapshot at 'starting_time_'.

- **model_state state_**
  State estimator.

- **Matrix<T, General, RowMajor> G_**
  Minimax gain.

- **bool is_model_error_variance_diagonal_**
  Is the model error variance composed of its diagonal part only?

- **Vector<T> D_tilde_inv_**
  Diagonal part of the model error.

- **int Nmode_Q_**
  Number of modes in the square root $Q^{1/2}$.

- **int model_error_description_**
  Form of the model error matrix $Q$: 0 = no model error, 1 = identity times a scalar.

- **T model_error_scalar_**
  In case, the model error is the identity multiplied by a scalar.

- **T bound_over_standard_deviation_**
  Conversion factor between standard deviation and bound.

- **Vector<T> e0_**
  Systematic error in the initial condition.

- **Vector<T> e_**
  Systematic error.

- **Matrix<T, General, RowMajor> R_inv_**
  Observation error.

- **T eta_**
  Systematic error in the observations.

- **OutputSaver output_saver_**
  Output saver.

### 48.44.1 Detailed Description

This class implements a reduced minimax filter.
48.44.2 Constructor & Destructor Documentation

48.44.2.1 Verdandi::ReducedMinimax::ReducedMinimax ( )

Main constructor.
Builds the driver and reads option keys in the configuration file.

Parameters

| in  | configuration_file | configuration file |

Definition at line 40 of file ReducedMinimax.cxx.

48.44.3 Member Function Documentation

48.44.3.1 void Verdandi::ReducedMinimax::ComputeTangentLinearModel ( Matrix<T, General, RowMajor> & M )

Computes the full matrix of the tangent linear model.

Parameters

| out | M | the tangent linear model |

Definition at line 830 of file ReducedMinimax.cxx.

48.44.3.2 void Verdandi::ReducedMinimax::FinalizeStep ( )

Finalizes a step for the model.
Finalizes the model.
Definition at line 367 of file ReducedMinimax.cxx.

48.44.3.3 int Verdandi::ReducedMinimax::GetMode ( ) const

Returns the current mode (snapshot recording or minimax calculations).
Returns
The mode: 0 for snapshot recording and 1 for minimax calculations.
Definition at line 1001 of file ReducedMinimax.cxx.

48.44.3.4 Model & Verdandi::ReducedMinimax::GetModel ( )

Returns the model.
Returns
The model.
Definition at line 1036 of file ReducedMinimax.cxx.

48.44.3.5 string Verdandi::ReducedMinimax::GetName ( ) const [virtual]

Returns the name of the class.
Returns

The name of the class.

Reimplemented from Verdandi::VerdandiBase.

Definition at line 1071 of file ReducedMinimax.cxx.

48.44.3.6 ObservationManager & Verdandi::ReducedMinimax::GetObservationManager ( )

Returns the observation manager.

Returns

The observation manager.

Definition at line 1048 of file ReducedMinimax.cxx.

48.44.3.7 OutputSaver & Verdandi::ReducedMinimax::GetOutputSaver ( )

Returns the output saver.

Returns

The output saver.

Definition at line 1060 of file ReducedMinimax.cxx.

48.44.3.8 Matrix< T, General, RowMajor > & Verdandi::ReducedMinimax::GetPreviousProjection ( )

Returns the previous projection matrix.

Returns

The previous projection matrix.

Definition at line 1025 of file ReducedMinimax.cxx.

48.44.3.9 Matrix< T, General, RowMajor > & Verdandi::ReducedMinimax::GetProjection ( )

Returns the current projection matrix.

Returns

The current projection matrix.

Definition at line 1013 of file ReducedMinimax.cxx.

48.44.3.10 bool Verdandi::ReducedMinimax::HasFinished ( )

Checks whether the model has finished.

Returns

True if no more data assimilation is required, false otherwise.

Definition at line 979 of file ReducedMinimax.cxx.
void Verdandi::ReducedMinimax::Initialize ( string configuration_file, bool initialize_model = true, bool initialize_observation_manager = true )

Initializes the driver.
Initializes the model and the observation manager.
Definition at line 71 of file ReducedMinimax.cxx.

void Verdandi::ReducedMinimax::Initialize ( VerdandiOps & configuration, bool initialize_model = true, bool initialize_observation_manager = true )

Initializes the driver.
Initializes the model and the observation manager.
Definition at line 85 of file ReducedMinimax.cxx.

void Verdandi::ReducedMinimax::InitializeStep ( )

Initializes a step.
Initializes a step for the model.
Definition at line 263 of file ReducedMinimax.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/ReducedMinimax.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/ReducedMinimax.cxx

48.45 Verdandi::ReducedOrderExtendedKalmanFilter Class Reference

This class implements a reduced order extended Kalman filter.

#include <ReducedOrderExtendedKalmanFilter.hxx>

Inheritance diagram for Verdandi::ReducedOrderExtendedKalmanFilter:

```
Verdandi::VerdandiBase
   `-- Verdandi::ReducedOrderExtendedKalmanFilter
```

Public Types

- typedef Model::state::value_type Ts
  Value type of the model state.
- typedef Model::state_error_variance_row model_state_error_variance_row
  Type of a row of the background error variance.
- typedef Model::state model_state
  Type of the model state vector.
- typedef Model::matrix_state_observation matrix_state_observation
  Type of the model/observation crossed matrix.
- typedef Model::state_error_variance model_state_error_variance
  Type of the background error variance.
- typedef Model::tangent_linear_operator model_tangent_linear_operator
  Type of the tangent linear model.
- typedef ObservationManager::tangent_linear_operator observation_tangent_linear_operator
  Type of the tangent linear observation operator.
- typedef ObservationManager::error_variance observation_error_variance
  Type of the observation error variance.
- typedef ObservationManager::tangent_linear_operator_row observation_tangent_linear_operator_row
  Type of a row of the tangent linear observation operator.
- typedef ObservationManager::observation observation
  Type of the observation vector.
- typedef Matrix<Ts, General, RowMajor> dense_matrix
  Type of dense matrix.
- typedef Model::state_error_variance_reduced model_state_error_variance_reduced
  Type of the reduced matrix $U$ in the $LUU^T$ decomposition of the background error covariance matrix.

Public Member Functions

- ReducedOrderExtendedKalmanFilter ()
  Main constructor.
- ~ReducedOrderExtendedKalmanFilter ()
  Destructor.
- void Initialize (VerdandiOps &configuration, bool initialize_model=true, bool initialize_observation_manager=true)
  Initializes the driver.
- void Initialize (string configuration_file, bool initialize_model=true, bool initialize_observation_manager=true)
  Initializes the driver.
- void InitializeStep ()
  Initializes a step for the extended Kalman filter.
- void Forward ()
  Performs a step forward with assimilation.
- void Analyze ()
  Computes an analysis.
- void Prediction ()
  Performs a forecast step.
- void FinalizeStep ()
  Finalizes a step for the model.
- void Finalize ()
  Finalizes the model.
- void PropagateCovarianceMatrix ()
  Computes Covariance.
- bool HasFinished ()
  Checks whether the model has finished.
- Model & GetModel ()
  Returns the model.
- ObservationManager & GetObservationManager ()
  Returns the observation manager.
- OutputSaver & GetOutputSaver ()
  Returns the output saver.
- string GetName () const
  Returns the name of the class.
- void Message (string message)
  Receives and handles a message.
Static Public Member Functions

- static void StaticMessage (void *object, string message)
  Receives and handles a message with a static method.

Protected Attributes

- Model model_
  Underlying model.
- ObservationManager observation_manager_
  Observation manager.
- dense_matrix L_
  Matrix L in the P SVD decomposition.
- dense_matrix U_
  Matrix U in the P SVD decomposition.
- int iteration_
  Iteration.
- string configuration_file_
  Path to the configuration file.
- string model_configuration_file_
  Path to the model configuration file.
- string observation_configuration_file_
  Path to the configuration file for the observation manager.
- map<string, bool> option_display_
  Display options.
- int Nstate_
  Dimension of the state.
- int Nobservation_
  Number of observations.
- int Nreduced_
  Dimension of the filtered state.
- bool analyze_first_step_
  Should an analysis be computed at the first step?
- string observation_error_variance_
  Indicates how R is stored (matrix, matrix_inverse, vector).
- double time_
  Current time.
- OutputSaver output_saver_
  Output saver.

48.45.1 Detailed Description

This class implements a reduced order extended Kalman filter.

48.45.2 Constructor & Destructor Documentation

48.45.2.1 Verdandi::ReducedOrderExtendedKalmanFilter::ReducedOrderExtendedKalmanFilter ( )

Main constructor.
Builds the driver and reads option keys in the configuration file.
Parameters

| in | configuration_file | configuration file. |

Definition at line 45 of file ReducedOrderExtendedKalmanFilter.cxx.

48.45.3 Member Function Documentation

48.45.3.1 void Verdandi::ReducedOrderExtendedKalmanFilter::Analyze ( )

Computes an analysis.
Whenever observations are available, it computes BLUE.
Definition at line 483 of file ReducedOrderExtendedKalmanFilter.cxx.

48.45.3.2 Model & Verdandi::ReducedOrderExtendedKalmanFilter::GetModel ( )

Returns the model.
Returns
  The model.
Definition at line 733 of file ReducedOrderExtendedKalmanFilter.cxx.

48.45.3.3 string Verdandi::ReducedOrderExtendedKalmanFilter::GetName ( ) const [virtual]

Returns the name of the class.
Returns
  The name of the class.
Reimplemented from Verdandi::VerdandiBase.
Definition at line 772 of file ReducedOrderExtendedKalmanFilter.cxx.

48.45.3.4 ObservationManager & Verdandi::ReducedOrderExtendedKalmanFilter::GetObservationManager ( )

Returns the observation manager.
Returns
  The observation manager..
Definition at line 746 of file ReducedOrderExtendedKalmanFilter.cxx.

48.45.3.5 OutputSaver & Verdandi::ReducedOrderExtendedKalmanFilter::GetOutputSaver ( )

Returns the output saver.
Returns
  The output saver.
Definition at line 759 of file ReducedOrderExtendedKalmanFilter.cxx.
48.45.3.6 bool Verdandi::ReducedOrderExtendedKalmanFilter::HasFinished()

Checks whether the model has finished.

Returns

True if no more data assimilation is required, false otherwise.

Definition at line 720 of file ReducedOrderExtendedKalmanFiltercxx.

48.45.3.7 void Verdandi::ReducedOrderExtendedKalmanFilter::Initialize(VerdandiOps & configuration, bool initialize_model = true, bool initialize_observation_manager = true)

Initializes the driver.

Initializes the model and the observation manager. Optionally computes the analysis of the first step.

Definition at line 107 of file ReducedOrderExtendedKalmanFiltercxx.

48.45.3.8 void Verdandi::ReducedOrderExtendedKalmanFilter::Initialize(string configuration_file, bool initialize_model = true, bool initialize_observation_manager = true)

Initializes the driver.

Initializes the model and the observation manager. Optionally computes the analysis of the first step.

Definition at line 93 of file ReducedOrderExtendedKalmanFiltercxx.

48.45.3.9 void Verdandi::ReducedOrderExtendedKalmanFilter::InitializeStep()

Initializes a step for the extended Kalman filter.

Initializes a step for the model.

Definition at line 407 of file ReducedOrderExtendedKalmanFiltercxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/ReducedOrderExtendedKalmanFilter.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/ReducedOrderExtendedKalmanFilter.cxx

48.46 Verdandi::ReducedOrderUnscentedKalmanFilter Class Reference

This class implements a reduced order unscented Kalman filter.

#include <ReducedOrderUnscentedKalmanFilter.hxx>

Inheritance diagram for Verdandi::ReducedOrderUnscentedKalmanFilter:

```
Verdandi::ReducedOrderUnscentedKalmanFilter
```

Public Types

- typedef Model::state::value_type Ts
Value type of the model state.

- typedef Model::state_error_variance_row model_state_error_variance_row
  Type of a row of the background error variance.
- typedef Model::state model_state
  Type of the model state vector.
- typedef Model::matrix_state_observation matrix_state_observation
  Type of the model/observation crossed matrix.
- typedef Model::state_error_variance model_state_error_variance
  Type of the background error variance.
- typedef Model::tangent_linear_operator model_tangent_linear_operator
  Type of the tangent linear model.
- typedef ObservationManager::observation::value_type To
  Value type of the observation vector.
- typedef ObservationManager::tangent_linear_operator observation_tangent_linear_operator
  Type of the tangent linear observation operator.
- typedef ObservationManager::error_variance observation_error_variance
  Type of the observation error variance.
- typedef ObservationManager::tangent_linear_operator_row observation_tangent_linear_operator_row
  Type of a row of the tangent linear observation operator.
- typedef ObservationManager::observation observation
  Type of the observation vector.
- typedef Vector<Ts, VectFull, MallocAlloc<Ts>> sigma_point
  Type of the sigma point vector.
- typedef Vector<model_state, Collection> state_collection
  Type of the state vector collection.
- typedef Vector<observation, Collection> observation_collection
  Type of the observation vector collection.
- typedef Matrix<Ts, General, RowMajor, MallocAlloc<Ts>> sigma_point_matrix
  Type of the sigma point matrix.
- typedef Model::state_error_variance_reduced model_state_error_variance_reduced
  Type of the reduced matrix $U$ in the $LUU^T$ decomposition of the background error covariance matrix.

Public Member Functions

- ReducedOrderUnscentedKalmanFilter()
  Main constructor.
- ~ReducedOrderUnscentedKalmanFilter()
  Destructor.
- void Initialize (string configuration_file, bool initialize_model=true, bool initialize_observation_manager=true)
  Initializes the driver.
- void Initialize (VerdandiOps &configuration, bool initialize_model=true, bool initialize_observation_manager=true)
  Initializes the driver.
- void InitializeStep()
  Performs a step for the unscented Kalman filter.
- void Forward()
  Performs a step forward with assimilation.
- void Analyze()
  Computes an analysis.
- void Prediction()
Performs a forecast step.

- void FinalizeStep ()
  Finalizes a step for the model.

- void Finalize ()
  Finalizes the model.

- bool HasFinished ()
  Checks whether the model has finished.

- Model & GetModel ()
  Returns the model.

- ObservationManager & GetObservationManager ()
  Returns the observation manager.

- OutputSaver & GetOutputSaver ()
  Returns the output saver.

- string GetName () const
  Returns the name of the class.

- void Message (string message)
  Receives and handles a message.

Static Public Member Functions

- static void StaticMessage (void ∗object, string message)
  Receives and handles a message with a static method.

Protected Attributes

- Model model_
  Underlying model.

- ObservationManager observation_manager_
  Observation manager.

- model_state_error_variance_reduced U_
  Matrix U in the P SVD decomposition.

- sigma_point_matrix U_inv_
  Inverse of matrix U.

- sigma_point_matrix C_
  Innovation covariance.

- int iteration_
  Iteration.

- string configuration_file_
  Path to the configuration file.

- string model_configuration_file_
  Path to the model configuration file.

- string observation_configuration_file_
  Path to the configuration file for the observation manager.

- map<string, bool> option_display_
  Display options.

- int Nstate_
  Dimension of the state.

- int Nreduced_
  Dimension of the filtered state.

- int Nobservation_
Number of observations.
- bool analyze_first_step_
  Should an analysis be computed at the first step?
- bool with_resampling_
  Should resampling be performed.
- string observation_error_variance_
  Indicates how R is stored (matrix, matrix_inverse, vector).
- string sigma_point_type_
  Choice of sigma-points.
- sigma_point_matrix I_
  Matrix of sigma-points.
- sigma_point_matrix I_trans_
  Matrix of sigma-points (transposed).
- model_state_error_variance X_i_
  \([X_n^{(*)}]\).
- model_state_error_variance X_i_trans_
  \([X_n^{(*)}]^t\).
- sigma_point D_alpha_
  Coefficient vector associated with sigma-points.
- sigma_point_matrix D_v_
  \(P_{\alpha}^{-1}\{V\} \).
- bool alpha_constant_
  Boolean to indicate if the coefficients alpha are constants.
- Ts alpha_
  alpha.
- int Nsigma_point_
  Number of sigma-points.
- OutputSaver output_saver_
  Output saver.

48.46.1 Detailed Description

This class implements a reduced order unscented Kalman filter.

48.46.2 Constructor & Destructor Documentation

48.46.2.1 Verdandi::ReducedOrderUnscentedKalmanFilter::ReducedOrderUnscentedKalmanFilter ( )

Main constructor.
Builds the driver and reads option keys in the configuration file.

Parameters

| in | configuration_file | configuration file. |

Definition at line 49 of file ReducedOrderUnscentedKalmanFilter.cxx.

48.46.3 Member Function Documentation
48.46.3.1 void Verdandi::ReducedOrderUnscentedKalmanFilter::Analyze()

Computes an analysis. Whenever observations are available, it computes BLUE.
Definition at line 724 of file ReducedOrderUnscentedKalmanFilter.cxx.

48.46.3.2 Model & Verdandi::ReducedOrderUnscentedKalmanFilter::GetModel()

Returns the model.
Definition at line 1090 of file ReducedOrderUnscentedKalmanFilter.cxx.

48.46.3.3 string Verdandi::ReducedOrderUnscentedKalmanFilter::GetName() const [virtual]

Returns the name of the class.
Reimplemented from Verdandi::VerdandiBase.
Definition at line 1129 of file ReducedOrderUnscentedKalmanFilter.cxx.

48.46.3.4 ObservationManager & Verdandi::ReducedOrderUnscentedKalmanFilter::GetObservationManager()

Returns the observation manager.
Definition at line 1103 of file ReducedOrderUnscentedKalmanFilter.cxx.

48.46.3.5 OutputSaver & Verdandi::ReducedOrderUnscentedKalmanFilter::GetOutputSaver()

Returns the output saver.
Definition at line 1116 of file ReducedOrderUnscentedKalmanFilter.cxx.

48.46.3.6 bool Verdandi::ReducedOrderUnscentedKalmanFilter::HasFinished()

Checks whether the model has finished.
Definition at line 1077 of file ReducedOrderUnscentedKalmanFilter.cxx.
48.46.3.7 void Verdandi::ReducedOrderUnscentedKalmanFilter::Initialize ( string configuration_file, bool initialize_model = true, bool initialize_observation_manager = true )

Initializes the driver.
Initializes the model and the observation manager. Optionally computes the analysis of the first step.
Definition at line 93 of file ReducedOrderUnscentedKalmanFilter.cxx.

48.46.3.8 void Verdandi::ReducedOrderUnscentedKalmanFilter::Initialize ( VerdandiOps & configuration, bool initialize_model = true, bool initialize_observation_manager = true )

Initializes the driver.
Initializes the model and the observation manager. Optionally computes the analysis of the first step.
Definition at line 106 of file ReducedOrderUnscentedKalmanFilter.cxx.

48.46.3.9 void Verdandi::ReducedOrderUnscentedKalmanFilter::InitializeStep ( )

Initializes a step for the unscented Kalman filter.
Initializes a step for the model.
Definition at line 386 of file ReducedOrderUnscentedKalmanFilter.cxx.

The documentation for this class was generated from the following files:

– /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/ReducedOrderUnscentedKalmanFilter.hxx
– /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/ReducedOrderUnscentedKalmanFilter.cxx

48.47 Verdandi::ShallowWater Class Reference

This class is a shallow-water model.
#include <ShallowWater.hxx>

Inheritance diagram for Verdandi::ShallowWater:

```
Verdandi::VerdandiBase
   |__ Verdandi::ShallowWater
```

Public Types

– typedef T value_type
  The numerical type (e.g., double).
– typedef T * pointer
  Pointer to the numerical type.
– typedef const T * const_pointer
  Const pointer to the numerical type.
– typedef T & reference
  Reference to the numerical type.
typedef const T & const_reference
Const reference to the numerical type.

typedef Matrix<T, General, RowSparse> state_error_variance
Type of the background error covariance matrix.

typedef Vector<T> state_error_variance_row
Type of a row of the background error variance.

typedef Vector<T> state
Type of the model state vector.

typedef Matrix<T, General, RowSparse> matrix_state_observation
Type of the model/observation crossed matrix.

typedef Vector<T> uncertain_parameter
Type of an uncertain parameter.

typedef Matrix<T, Symmetric, Seldon::RowSymPacked> parameter_variance
Type of the uncertain parameter covariance matrix.

Public Member Functions

ShallowWater()
Constructor.

ShallowWater (string configuration_file)
Constructor.

∼ShallowWater()
Destructor.

void Initialize (string configuration_file)
Initializes the model.

void InitializeStep()
Initializes the current time step for the model.

void Forward()
Advances one step forward in time.

bool HasFinished () const
Checks whether the model has finished.

void Save()
Saves the simulated data.

void FinalizeStep()
Finalizes the current time step for the model.

void Finalize()
Finalizes the model.

double GetTime () const
Returns the current time.

void SetTime (double time)
Sets the current time.

int GetNx () const
Returns the number of points along x (in the grid for height).

int GetNy () const
Returns the number of points along y (in the grid for height).

int GetNz () const
Returns the number of points along z.

int GetXMin () const
Returns the first abscissa.

int GetYMin () const
Verdandi::ShallowWater Class Reference

Returns the first ordinate.

- int GetDeltaX () const
  Returns the space step along x.
- int GetDeltaY () const
  Returns the space step along y.
- int GetNstate () const
  Returns the number of points in the grid.
- int GetNfull_state () const
  Returns the full state vector size.
- state & GetState ()
  Provides the state vector.
- void StateUpdated ()
  Performs some calculations when the update of the model state is done.
- state & GetFullState ()
  Provides the full state vector.
- void FullStateUpdated ()
  Performs some calculations when the update of the model state is done.
- state_error_variance_row & GetStateErrorVarianceRow (int row)
  Computes a row of the background error covariance matrix B.
- const state_error_variance & GetStateErrorVariance () const
  Returns the background error covariance matrix (B) if available.
- bool IsErrorSparse () const
  Checks if the error covariance matrix is sparse.
- int GetNparameter ()
  Returns the number of parameters to be perturbed.
- uncertain_parameter & GetParameter (int i)
  Gets the i-th uncertain parameter.
- void ParameterUpdated (int i)
- Vector<T> & GetParameterCorrelation (int i)
  Returns the correlation between the uncertain parameters.
- string GetParameterPDF (int i)
  Returns the PDF of the i-th parameter.
- parameter_variance & GetParameterVariance (int i)
  Returns the covariance matrix associated with the i-th parameter.
- Vector<T> & GetParameterPDFData (int i)
  Returns parameters associated with the PDF of some model parameter.
- string GetParameterOption (int i)
  Returns the perturbation option of the i-th parameter.
- const ShallowWater<T> & GetModel () const
  Returns the model itself.
- string GetName () const
  Returns the name of the class.
- void Message (string message)
  Receives and handles a message.

Static Public Member Functions

- static void StaticMessage (void *object, string message)
  Receives and handles a message with a static method.
Protected Attributes

- Vector< T > state_
  State vector.
- Vector< T > full_state_
  Full state vector.
- Matrix< T > h_
  Water height.
- Matrix< T > u_
  Vertical velocity along x.
- Matrix< T > v_
  Vertical velocity along y.
- Matrix< T > hf_x_
  Water-height flux along x.
- Matrix< T > hf_y_
  Water-height flux along y.
- Matrix< T > huf_x_
  Flux along x of the flow rate along x.
- Matrix< T > huf_y_
  Flux along y of the flow rate along x.
- Matrix< T > hvf_x_
  Flux along x of the flow rate along y.
- Matrix< T > hvf_y_
  Flux along y of the flow rate along y.
- double x_min_
  First abscissa.
- double y_min_
  First ordinate.
- double Delta_x_
  Space step along x.
- double Delta_y_
  Space step along y.
- int Nx_
  Number of points along x (in the grid for height).
- int Ny_
  Number of points along y (in the grid for height).
- double Delta_t_
  Time step.
- double time_
  Current time.
- double final_time_
  Simulation duration.
- const double g_
  Gravitational acceleration.
- int boundary_condition_left_
  Type of boundary condition on the left (0: free, 1: wall, 2: flow, 3: height).
- T value_left_
  Constant in-flow or height on the left.
- T amplitude_left_
  Amplitude of variations on the left.
- T frequency_left_
Frequency of the variations on the left.

- int `boundary_condition_right_`
  Type of boundary condition on the right (0: free, 1: wall, 2: flow, 3: height).

- T `value_right_`
  Constant in-flow or height on the right.

- T `amplitude_right_`
  Amplitude of variations on the right.

- T `frequency_right_`
  Frequency of the variations on the right.

- int `boundary_condition_bottom_`
  Type of boundary condition on the bottom (0: free, 1: wall, 2: flow, 3: height).

- T `value_bottom_`
  Constant in-flow or height on the bottom.

- T `amplitude_bottom_`
  Amplitude of variations on the bottom.

- T `frequency_bottom_`
  Frequency of the variations on the bottom.

- int `boundary_condition_top_`
  Type of boundary condition on the top (0: free, 1: wall, 2: flow, 3: height).

- T `value_top_`
  Constant in-flow or height on the top.

- T `amplitude_top_`
  Amplitude of variations on the top.

- T `frequency_top_`
  Frequency of the variations on the top.

- T `value_`
  Value of the departure in the initial conditions.

- bool `source_center_`
  Is there an initial step at the center?

- bool `source_left_`
  Is there an initial step on the left?

- double `model_error_std_bc_`
  Standard deviation of the model error for boundary conditions.

- double `model_error_std_ic_`
  Standard deviation of the model error for initial conditions.

- string `seed_`
  Determining the random seed.

- std::tr1::mt19937 `generator_`
  Generates random numbers, using Mersenne twister.

- double `Balgovind_scale_background_`
  Balgovind scale for background covariance.

- double `state_error_variance_value_`
  Background error variance.

- state_error_variance `state_error_variance_`
  Background error covariance matrix (B).

- double `Balgovind_scale_model_`
  Balgovind scale for model covariance.

- double `model_error_variance_`
  Model error variance.

- int `current_row_`
  Number of the row of B currently stored.
- `int current_column_`
  
  *Number of the column of Q currently stored.*

- `state_error_variance_row state_error_variance_row_`
  
  *Value of the row of B currently stored.*

- `bool with_positivity_requirement_`
  
  *Flag that indicates whether the positivity of the analyzed data is required.*

- `vector< uncertain_parameter > parameter_`
  
  *Parameters to be perturbed.*

- `vector< string > uncertain_parameter_vector_`
  
  *List of parameters to be perturbed.*

- `int Nparameter_`
  
  *Number of parameters to be perturbed.*

- `Vector< T > step_height_parameter_`
  
  *PDF parameters for the step height.*

- `Vector< T > step_height_correlation_`
  
  *Correlations between the step height and the other parameters.*

- `string step_height_pdf_`
  
  *Name of the probability distribution for the step height.*

- `T step_height_mean_`
  
  *Mean of the probability distribution for the step height.*

- `Matrix< T, Symmetric, RowSymPacked > step_height_variance_`
  
  *Covariance matrix for the step height.*

- `Vector< T > bc_parameter_`
  
  *PDF parameters for the boundary condition.*

- `Vector< T > bc_correlation_`
  
  *Correlations between the boundary condition and the other parameters.*

- `string bc_pdf_`
  
  *Name of the probability distribution for the boundary condition.*

- `T bc_mean_`
  
  *Mean of the probability distribution for the boundary condition.*

- `Matrix< T, Symmetric, RowSymPacked > bc_variance_`
  
  *Covariance matrix for the boundary condition.*

- `OutputSaver output_saver_`
  
  *Output saver.*

### 48.47.1 Detailed Description

This class is a shallow-water model.

**Template Parameters**

| T | the type of floating-point numbers. |

### 48.47.2 Constructor & Destructor Documentation

#### 48.47.2.1 Verdandi::<ShallowWater>::ShallowWater ( string configuration_file )

Constructor.

It builds allocates the state vectors.
Parameters

| in | configuration_file | path to the configuration file. |

Definition at line 61 of file ShallowWater.cxx.

48.47.3 Member Function Documentation

48.47.3.1 int Verdandi::ShallowWater::GetDeltaX ( ) const

Returns the space step along x.

Returns

The space step along x.

Definition at line 673 of file ShallowWater.cxx.

48.47.3.2 int Verdandi::ShallowWater::GetDeltaY ( ) const

Returns the space step along y.

Returns

The space step along y.

Definition at line 684 of file ShallowWater.cxx.

48.47.3.3 ShallowWater<T>::state & Verdandi::ShallowWater::GetFullState ( )

Provides the full state vector.

Parameters

| out | state | the full state vector. |

Definition at line 752 of file ShallowWater.cxx.

48.47.3.4 const ShallowWater<T> & Verdandi::ShallowWater::GetModel ( ) const

Returns the model itself.

Returns

The model.

Definition at line 1000 of file ShallowWater.cxx.

48.47.3.5 string Verdandi::ShallowWater::GetName ( ) const [virtual]

Returns the name of the class.
Returns

The name of the class.

Reimplemented from Verdandi::VerdandiBase.
Definition at line 1011 of file ShallowWater.cxx.

48.47.3.6 int Verdandi::ShallowWater::GetNfull_state ( ) const

Returns the full state vector size.

Returns

The full state vector size.

Definition at line 706 of file ShallowWater.cxx.

48.47.3.7 int Verdandi::ShallowWater::GetNparameter ( )

Returns the number of parameters to be perturbed.

Returns

The number of parameters to be perturbed.

Definition at line 883 of file ShallowWater.cxx.

48.47.3.8 int Verdandi::ShallowWater::GetNstate ( ) const

Returns the number of points in the grid.

Returns

The number of points in the grid.

Definition at line 695 of file ShallowWater.cxx.

48.47.3.9 int Verdandi::ShallowWater::GetNx ( ) const

Returns the number of points along x (in the grid for height).

Returns

The number of points along x (in the grid for height).

Definition at line 618 of file ShallowWater.cxx.

48.47.3.10 int Verdandi::ShallowWater::GetNy ( ) const

Returns the number of points along y (in the grid for height).

Returns

The number of points along y (in the grid for height).

Definition at line 629 of file ShallowWater.cxx.
48.47.3.11 int Verdandi::ShallowWater::GetNz () const

Returns the number of points along z.

Returns

The number of points along z.

Definition at line 640 of file ShallowWater.cxx.

48.47.3.12 ShallowWater< T >::uncertain_parameter & Verdandi::ShallowWater::GetParameter ( int k )

Gets the i-th uncertain parameter.

Parameters

| in | i | index of the parameter. |

Returns

The vector associated with the i-th parameter.

Definition at line 896 of file ShallowWater.cxx.

48.47.3.13 Vector< T > & Verdandi::ShallowWater::GetParameterCorrelation ( int i )

Returns the correlation between the uncertain parameters.
Since there is only one parameter, an empty vector is returned.

Parameters

| in | i | parameter index. |

Returns

An empty vector.

Definition at line 921 of file ShallowWater.cxx.

48.47.3.14 string Verdandi::ShallowWater::GetParameterOption ( int i )

Returns the perturbation option of the i-th parameter.

Parameters

| in | i | parameter index. |

Returns

The perturbation option of the i-th parameter.

Definition at line 986 of file ShallowWater.cxx.
48.47.3.15  string Verdandi::ShallowWater::GetParameterPDF ( int \textit{i} )

Returns the PDF of the \textit{i}-th parameter.

**Parameters**

\begin{tabular}{|c|}
\hline
\texttt{in} \textit{i} \text{ uncertain-variable index.} \\
\hline
\end{tabular}

**Returns**

The PDF of the \textit{i}-th parameter.

Definition at line 936 of file ShallowWater.cxx.

48.47.3.16  Vector< T> & Verdandi::ShallowWater::GetParameterPDFData ( int \textit{i} )

Returns parameters associated with the PDF of some model parameter. In case of normal or log-normal distribution, the parameters are clipping parameters.

**Parameters**

\begin{tabular}{|c|}
\hline
\texttt{in} \textit{i} \text{ model parameter index.} \\
\hline
\end{tabular}

**Returns**

The parameters associated with the \textit{i}-th parameter.

Definition at line 970 of file ShallowWater.cxx.

48.47.3.17  ShallowWater< T>::parameter_variance & Verdandi::ShallowWater::GetParameterVariance ( int \textit{i} )

Returns the covariance matrix associated with the \textit{i}-th parameter.

**Parameters**

\begin{tabular}{|c|}
\hline
\texttt{in} \textit{i} \text{ parameter index.} \\
\hline
\end{tabular}

**Returns**

The covariance matrix associated with the \textit{i}-th parameter.

Definition at line 953 of file ShallowWater.cxx.

48.47.3.18  ShallowWater< T>::state & Verdandi::ShallowWater::GetState ( )

Provides the state vector.

**Returns**

A reference to the state vector.

Definition at line 718 of file ShallowWater.cxx.

48.47.3.19  const ShallowWater< T>::state_error_variance & Verdandi::ShallowWater::GetStateErrorVariance ( ) const

Returns the background error covariance matrix (\textit{B}) if available.
Returns the background error covariance matrix (B) if available, raises an exception otherwise.

Returns

The matrix of the background error covariance.

Definition at line 851 of file ShallowWater.cxx.

48.47.3.20 ShallowWater< T >::state_error_variance_row & Verdandi::ShallowWater::GetStateErrorVarianceRow ( int row )

Computes a row of the background error covariance matrix B.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>row</th>
<th>row index</th>
</tr>
</thead>
</table>

Returns

The value of row number row.

Definition at line 788 of file ShallowWater.cxx.

48.47.3.21 double Verdandi::ShallowWater::GetTime ( ) const

Returns the current time.

Returns

The current time.

Definition at line 596 of file ShallowWater.cxx.

48.47.3.22 int Verdandi::ShallowWater::GetXMin ( ) const

Returns the first abscissa.

Returns

The first abscissa.

Definition at line 651 of file ShallowWater.cxx.

48.47.3.23 int Verdandi::ShallowWater::GetYMin ( ) const

Returns the first ordinate.

Returns

The first ordinate.

Definition at line 662 of file ShallowWater.cxx.
48.47.3.24  bool Verdandi::ShallowWater::HasFinished ( ) const

Checks whether the model has finished.

Returns

True if no more data assimilation is required, false otherwise.

Definition at line 554 of file ShallowWater.cxx.

48.47.3.25  void Verdandi::ShallowWater::Initialize ( string configuration_file )

Initializes the model.

Parameters

| in    | configuration_file | configuration file. |

Definition at line 101 of file ShallowWater.cxx.

48.47.3.26  bool Verdandi::ShallowWater::IsErrorSparse ( ) const

Checks if the error covariance matrix is sparse.

Returns

True if there is a sparse error matrix, false otherwise.

Definition at line 868 of file ShallowWater.cxx.

48.47.3.27  void Verdandi::ShallowWater::ParameterUpdated ( int i )

Performs some calculations when the update of the i-th uncertain parameter is done.

Parameters

| in    | i | index of the parameter. |

Definition at line 909 of file ShallowWater.cxx.

48.47.3.28  void Verdandi::ShallowWater::Save ( )

Saves the simulated data.

It saves the velocities 'u' and 'v', and the height 'h'.

Definition at line 564 of file ShallowWater.cxx.

48.47.3.29  void Verdandi::ShallowWater::SetTime ( double time )

Sets the current time.

Parameters

| in    | time | the current time. |

Definition at line 607 of file ShallowWater.cxx.
The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/model/ShallowWater.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/model/ShallowWater.cxx

### 48.48 Verdandi::TR1PerturbationManager Class Reference

This class generates random samples using C++ TR1 library.

```cpp
#include <TR1PerturbationManager.hxx>
```

Inheritance diagram for Verdandi::TR1PerturbationManager:

```
Verdandi::BasePerturbationManager< TR1PerturbationManager >
Verdandi::TR1PerturbationManager
```

#### Public Member Functions

- `TR1PerturbationManager ()`
  *Default constructor.*
- `TR1PerturbationManager (string configuration_file)`
  *Main constructor.*
- `TR1PerturbationManager (const TR1PerturbationManager &)`
  *Constructor by copy.*
- `~TR1PerturbationManager ()`
  *Destructor.*
- `void Initialize (string configuration_file)`
  *Initializes the manager.*
- `void Initialize (VerdandiOps &configuration_stream)`
  *Initializes the manager.*
- `void Finalize ()`
  *Does nothing.*
- `double Normal (double mean, double variance, Vector< double, VectFull > &parameter)`
  *Generates a random number with a normal distribution.*
- `double LogNormal (double mean, double variance, Vector< double, VectFull > &parameter)`
  *Generates a random number with a log-normal distribution.*
- `double Uniform (double min, double max)`
  *Generates a random number from a uniform distribution.*
- `int UniformInt (int min, int max)`
  *Generates a random number from a uniform distribution on integers.*
- `template<class T0 , class Prop0 , class Allocator0 > void Normal (Matrix< T0, Prop0, RowSymPacked, Allocator0 > &variance, Vector< double, VectFull > &parameter, Vector< T1, VectFull, Allocator0 > &sample)`
  *Generates a vector of random numbers with a normal distribution.*
- `template<class T0 , class Prop0 , class Allocator0 , class T1 , class AllocVector0 > void LogNormal (Matrix< T0, Prop0, RowSymPacked, Allocator0 > &variance, Vector< double, VectFull > &parameter, Vector< T1, VectFull, AllocVector0 > &output)`
  *Generate a random vector with a log-normal distribution.*
– template<class T0 , class T1 , class Allocator1 >
  void NormalHomogeneous (T0 variance, Vector< double, VectFull > &parameter, Vector< T1, VectFull, -Allocator1 > &output)

  Generate a random vector with a homogeneous normal distribution.

– template<class T0 , class T1 , class Allocator1 >
  void LogNormalHomogeneous (T0 variance, Vector< double, VectFull > &parameter, Vector< T1, VectFull, Allocator1 > &output)

  Generates a random vector with a homogeneous log normal distribution.

– template<class T0 , class T1 , class Allocator0 >
  bool NormalClipping (Vector< T0, VectFull > &diagonal, Vector< double, VectFull > &parameter, Vector< T1, VectFull, Allocator0 > &output)

  Tests if a vector satisfies clipping constraints.

– template<class T0 , class T1 , class Allocator1 >
  bool NormalClipping (Vector< T0, VectFull > &diagonal, Vector< double, VectFull > &parameter, Vector< T1, VectFull, Allocator1 > &output)

  – void Sample (string pdf, Matrix< T0, Prop0, RowSymPacked, Allocator0 > &variance, Vector< double, VectFull > &parameter, Vector< double, VectFull > &correlation, Vector< T1, VectFull, Allocator1 > &output)

  – void Sample (string pdf, Matrix< T0, Prop0, RowSymPacked, Allocator0 > &variance, Vector< double, VectFull > &parameter, Vector< double, VectFull > &correlation, Vector< T1, Collection, Allocator1 > &output)

Protected Types

– typedef tr1::mt19937 engine
– typedef tr1::uniform_real< double > distribution_uniform
– typedef tr1::variate_generator< engine, distribution_uniform > generator_uniform
– typedef tr1::normal_distribution< double > distribution_normal
– typedef tr1::variate_generator< engine, distribution_normal > generator_normal

Protected Attributes

– engine * urng_
  Mersenne Twister random number generator.
– distribution_uniform * distribution_uniform_
  Uniform distribution.
– generator_uniform * variate_generator_uniform_
  Uniform variate generator.
– distribution_normal * distribution_normal_
  Uniform distribution.
– generator_normal * variate_generator_normal_
  Uniform variate generator.

48.48.1 Detailed Description

This class generates random samples using C++ TR1 library.
48.48.2 Constructor & Destructor Documentation

48.48.2.1 Verdandi::TR1PerturbationManager::TR1PerturbationManager ( )

Default constructor.
The seed is initialized from the system clock.
Definition at line 44 of file TR1PerturbationManager.cxx.

48.48.2.2 Verdandi::TR1PerturbationManager::TR1PerturbationManager ( string configuration_file )

Main constructor.
Builds the manager and reads option keys in the configuration file.

Parameters

| in     | configuration_file | configuration file. |

Definition at line 63 of file TR1PerturbationManager.cxx.

48.48.2.3 Verdandi::TR1PerturbationManager::TR1PerturbationManager ( const TR1PerturbationManager & unUsed )

Constructor by copy.
The seed is initialized from the system clock.
Definition at line 81 of file TR1PerturbationManager.cxx.

48.48.3 Member Function Documentation

48.48.3.1 void Verdandi::TR1PerturbationManager::Initialize ( string configuration_file )

Initializes the manager.

Parameters

| in     | configuration_file | configuration file. |

Reimplemented from Verdandi::BasePerturbationManager< TR1PerturbationManager >.
Definition at line 115 of file TR1PerturbationManager.cxx.

48.48.3.2 void Verdandi::TR1PerturbationManager::Initialize ( VerdandiOps & configuration_stream )

Initializes the manager.

Parameters

| in     | configuration_stream | configuration stream. |

Definition at line 126 of file TR1PerturbationManager.cxx.
Generates a random number with a log-normal distribution.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>mean</th>
<th>mean of the distribution.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>variance</td>
<td>variance of the random variable.</td>
</tr>
<tr>
<td>in</td>
<td>parameter</td>
<td>vector of parameters. The vector may either be empty or contain two clipping parameters ((a, b)). With the clipping parameters, for a normal distribution, any random value lies in ([\mu - a\sigma, \mu + b\sigma]) where (\mu) is the mean of the random variable and (\sigma) is its standard deviation.</td>
</tr>
</tbody>
</table>

**Returns**

A random number following the previously described normal distribution.

Definition at line 187 of file TR1PerturbationManager.cxx.

Generates a random number with a normal distribution.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>mean</th>
<th>mean of the distribution.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>variance</td>
<td>variance of the random variable.</td>
</tr>
<tr>
<td>in</td>
<td>parameter</td>
<td>vector of parameters. The vector may either be empty or contain two clipping parameters ((a, b)). With the clipping parameters, for a normal distribution, any random value lies in ([\mu - a\sigma, \mu + b\sigma]) where (\mu) is the mean of the random variable and (\sigma) is its standard deviation.</td>
</tr>
</tbody>
</table>

**Returns**

A random number following the previously described normal distribution.

Definition at line 153 of file TR1PerturbationManager.cxx.

Generates a vector of random numbers with a normal distribution. Each component of the random vector is generated independently.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>variance</th>
<th>variance of the random variable.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>parameter</td>
<td>vector of parameters. The vector may either be empty or contain two clipping parameters ((a, b)). With the clipping parameters, for a normal distribution, any random value lies in ([\mu - a\sigma, \mu + b\sigma]) where (\mu) is the mean of the random variable and (\sigma) is its standard deviation.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>the generated random vector.</td>
</tr>
</tbody>
</table>
Definition at line 244 of file TR1PerturbationManager.cxx.

48.48.3.6 template< class T0 , class T1 , class Allocator1 > bool Verdandi::TR1PerturbationManager::NormalClipping ( Vector< T0, VectFull >& diagonal, Vector< double, VectFull >& parameter, Vector< T1, VectFull, Allocator1 >& output )

Tests if a vector satisfies clipping constraints.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>diagonal</th>
<th>diagonal coefficients of the covariance matrix.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>permutation</td>
<td>vector of permutations done during the Cholesky decomposition.</td>
</tr>
<tr>
<td></td>
<td>parameter</td>
<td>vector of parameters. The vector may either be empty or contain two clipping parameters ((a, b)). With the clipping parameters, for a normal distribution, any random value lies in ([\mu - a\sigma, \mu + b\sigma]) where (\mu) is the mean of the random variable and (\sigma) is its standard deviation.</td>
</tr>
<tr>
<td></td>
<td>output</td>
<td>vector to be tested. This vector was generated using a covariance matrix with diagonal diagonal.</td>
</tr>
</tbody>
</table>

Returns

true if the vector satisfies the constraints.

Definition at line 394 of file TR1PerturbationManager.cxx.

48.48.3.7 double Verdandi::TR1PerturbationManager::Uniform ( double min, double max )

Generates a random number from a uniform distribution.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>min</th>
<th>lower bound of the distribution.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>max</td>
<td>upper bound of the distribution.</td>
</tr>
</tbody>
</table>

Returns

A random number sampled from the uniform distribution \(U(min, max)\).

Definition at line 208 of file TR1PerturbationManager.cxx.

48.48.3.8 int Verdandi::TR1PerturbationManager::UniformInt ( int min, int max )

Generates a random number from a uniform distribution on integers.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>min</th>
<th>lower bound of the distribution.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>max</td>
<td>upper bound of the distribution.</td>
</tr>
</tbody>
</table>

Returns

A random integer sampled from the uniform distribution \(U(min, max)\).

Definition at line 224 of file TR1PerturbationManager.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/TR1PerturbationManager.hxx

Generated on Mon Sep 28 2015 16:49:58 for Verdandi by Doxygen
Verdandi::TrajectoryManager Class Reference

This class manages model trajectory.

```
#include <TrajectoryManager.hxx>
```

Public Types

- `typedef Model::state::value_type Ts`
  Value type of the model state.
- `typedef Model::state model_state`
  Type of the model state vector.

Public Member Functions

- `TrajectoryManager ()`
  Main constructor.
- `~TrajectoryManager ()`
  Destructor.
- `void Initialize (string checkpoint_recording_mode="memory", string checkpoint_recording_file="", string loaded_trajectory_recording_mode="memory", string loaded_trajectory_recording_file="", int Nskip_checkpoint=1)`
  Initializes the trajectory manager.
- `void Save (model_state &state, double time)`
  It appends state at a given time to the checkpoints.
- `void SetTime (Model &model, double time)`
  Sets the time of states to be loaded.
- `model_state & GetState ()`
  Returns the state at current time.
- `double GetTime ()`
  Returns the current time.
- `void Deallocate ()`
  Clears inner vector collection.
- `void EmptyFile (string file_name)`
  Empties the recording file.

Protected Attributes

- `string checkpoint_recording_mode_`
  Checkpoint recording mode.
- `string checkpoint_recording_file_`
  Checkpoint recording file name.
- `string loaded_trajectory_recording_mode_`
  Loaded trajectory recording mode.
- `string loaded_trajectory_recording_file_`
  Loaded trajectory aving file name.
- `int Nsave_call_`
  Number of call to save method.
48.49 Verdandi::TrajectoryManager Class Reference

- int Nstate_
  Model Nstate.
- int Nskip_checkpoint_
  Recording period.
- int Ncheckpoint_
  Number of recording in first forward loop.
- vector< model_state > checkpoint_
  Trajectory saved in first forward loop.
- Vector< double > checkpoint_time_
  Times associated with checkpoint_.
- vector< model_state > loaded_trajectory_
  Trajectory saved in backward loop.
- Vector< double > loaded_time_
  Times associated with loaded_trajectory_.
- int checkpoint_index_
  Checkpoint index.
- model_state input_data_
  State loaded from file.
- int loaded_trajectory_index_
  Loaded trajectory index.

48.49.1 Detailed Description

This class manages model trajectory.

48.49.2 Member Function Documentation

48.49.2.1 void Verdandi::TrajectoryManager::EmptyFile ( string file_name )

Empties the recording file.

Parameters

| in    | file_name | path to the file to be emptied. |

Definition at line 356 of file TrajectoryManager.cxx.

48.49.2.2 TrajectoryManager< Model >::model_state & Verdandi::TrajectoryManager::GetState ( )

Returns the state at current time.

Returns

The current state.

Definition at line 294 of file TrajectoryManager.cxx.

48.49.2.3 double Verdandi::TrajectoryManager::GetTime ( )

Returns the current time.
Returns

The current time.

Definition at line 326 of file TrajectoryManager.cxx.

48.49.2.4 void Verdandi::TrajectoryManager::Save ( model_state & state, double time )

It appends state at a given time time to the checkpoints.

Parameters

<table>
<thead>
<tr>
<th>state</th>
<th>a given state.</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>the given time.</td>
</tr>
</tbody>
</table>

Definition at line 101 of file TrajectoryManager.cxx.

48.49.2.5 void Verdandi::TrajectoryManager::SetTime ( Model & model, double time )

Sets the time of states to be loaded.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>model</th>
<th>the model.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>time</td>
<td>a given time.</td>
</tr>
</tbody>
</table>

Definition at line 138 of file TrajectoryManager.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/TrajectoryManager.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/TrajectoryManager.cxx

48.50 Verdandi::TRNGPerturbationManager Class Reference

This class generates random samples using TRNG.

#include <TRNGPerturbationManager.hxx>

Inheritance diagram for Verdandi::TRNGPerturbationManager:

```
Verdandi::BasePerturbationManager< TRNGPerturbationManager >
```

```
Verdandi::TRNGPerturbationManager
```

Public Member Functions

- **TRNGPerturbationManager ()**
  Default constructor.

- **TRNGPerturbationManager (string configuration_file)**
  Main constructor.

- **~TRNGPerturbationManager ()**
  Destructor.
void Initialize (string configuration_file)
Initialize the manager.

void Initialize (VerdandiOps &configuration_stream)
Initialize the manager.

void Finalize()
Does nothing.

double Normal (double mean, double variance, Vector<double, VectFull> &parameter)
Generates a random number with a normal distribution.

double LogNormal (double mean, double variance, Vector<double, VectFull> &parameter)
Generates a random number with a log-normal distribution.

template<class T0, class T1, class Prop0, class Allocator0>
void Normal (Matrix<T0, Prop0, RowSymPacked, Allocator0> variance, Vector<double, VectFull> &parameter, Vector<T1, VectFull, Allocator0> &sample)
Generates a vector of random numbers with a normal distribution.

template<class T0, class Prop0, class Allocator0, class T1, class Allocator1>
void LogNormal (Matrix<T0, Prop0, RowSymPacked, Allocator0> variance, Vector<double, VectFull> &parameter, Vector<T1, VectFull, Allocator1> &output)
Generate a random vector with a log-normal distribution.

template<class T0, class T1, class Allocator1>
void NormalHomogeneous (T0 variance, Vector<double, VectFull> &parameter, Vector<T1, VectFull, Allocator1> &output)
Generate a random vector with a homogeneous normal distribution.

template<class T0, class T1, class Allocator1>
void LogNormalHomogeneous (T0 variance, Vector<double, VectFull> &parameter, Vector<T1, VectFull, Allocator1> &output)
Generates a random vector with a homogeneous log normal distribution.

template<class T0, class T1, class Allocator0>
bool NormalClipping (Vector<T0, VectFull> &diagonal, Vector<double, VectFull> &parameter, Vector<T1, VectFull, Allocator0> &output)
Tests if a vector satisfies clipping constraints.

void Sample (string pdf, Matrix<T0, Prop0, RowSymPacked, Allocator0> &variance, Vector<double, VectFull> &parameter, Vector<double, VectFull> &correlation, Vector<T1, VectFull, Allocator1> &output)

void Sample (string pdf, Matrix<T0, Prop0, RowSymPacked, Allocator0> &variance, Vector<double, VectFull> &parameter, Vector<double, VectFull> &correlation, Vector<T1, VectFull, Allocator1> &output)

Protected Attributes

trng::yarn5 * nrng_
Normal random number generator.

48.50.1 Detailed Description

This class generates random samples using TRNG.
48.50.2 Constructor & Destructor Documentation

48.50.2.1 Verdandi::TRNGPerturbationManager::TRNGPerturbationManager ( )

Default constructor.
The seed is initialized from the system clock.
Definition at line 47 of file TRNGPerturbationManager.cxx.

48.50.2.2 Verdandi::TRNGPerturbationManager::TRNGPerturbationManager ( string configuration_file )

Main constructor.
Builds the manager and reads option keys in the configuration file.
Parameters

| in  | configuration_file | configuration file. |

Definition at line 59 of file TRNGPerturbationManager.cxx.

48.50.3 Member Function Documentation

48.50.3.1 void Verdandi::TRNGPerturbationManager::Initialize ( string configuration_file )

Initializes the manager.
Parameters

| in  | configuration_file | configuration file. |

Reimplemented from Verdandi::BasePerturbationManager< TRNGPerturbationManager >.
Definition at line 83 of file TRNGPerturbationManager.cxx.

48.50.3.2 void Verdandi::TRNGPerturbationManager::Initialize ( VerdandiOps & configuration_stream )

Initializes the manager.
Parameters

| in  | configuration_stream | configuration stream. |

Definition at line 94 of file TRNGPerturbationManager.cxx.

48.50.3.3 double Verdandi::TRNGPerturbationManager::LogNormal ( double mean, double variance, Vector< double, VectFull > & parameter )

Generates a random number with a log-normal distribution.
### Class Reference: TRNGPerturbationManager

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>mean</th>
<th>mean of the distribution.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>variance</td>
<td>variance of the random variable.</td>
</tr>
<tr>
<td>in</td>
<td>parameter</td>
<td>vector of parameters. The vector may either be empty or contain two clipping parameters ((a, b)). With the clipping parameters, for a normal distribution, any random value lies in ([\mu - a\sigma, \mu + b\sigma]) where (\mu) is the mean of the random variable and (\sigma) is its standard deviation.</td>
</tr>
</tbody>
</table>

#### Returns

A random number following the previously described normal distribution.

Definition at line 158 of file TRNGPerturbationManager.hxx.

#### Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>mean</th>
<th>mean of the distribution.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>variance</td>
<td>variance of the random variable.</td>
</tr>
<tr>
<td>in</td>
<td>parameter</td>
<td>vector of parameters. The vector may either be empty or contain two clipping parameters ((a, b)). With the clipping parameters, for a normal distribution, any random value lies in ([\mu - a\sigma, \mu + b\sigma]) where (\mu) is the mean of the random variable and (\sigma) is its standard deviation.</td>
</tr>
</tbody>
</table>

#### Returns

A random number following the previously described normal distribution.

Definition at line 121 of file TRNGPerturbationManager.hxx.

#### Parameters

| template<class T0, class T1, class Prop0, class Allocator0> void Verdandi::TRNGPerturbationManager::Normal ( double mean, double variance, Vector< double, VectFull > & parameter ) |

Generates a vector of random numbers with a normal distribution. Each component of the random vector is generated independently.

<table>
<thead>
<tr>
<th>in</th>
<th>variance</th>
<th>variance of the random variable.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>parameter</td>
<td>vector of parameters. The vector may either be empty or contain two clipping parameters ((a, b)). With the clipping parameters, for a normal distribution, any random value lies in ([\mu - a\sigma, \mu + b\sigma]) where (\mu) is the mean of the random variable and (\sigma) is its standard deviation.</td>
</tr>
<tr>
<td>out</td>
<td>output</td>
<td>the generated random vector.</td>
</tr>
</tbody>
</table>

Definition at line 192 of file TRNGPerturbationManager.hxx.
Tests if a vector satisfies clipping constraints.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>diagonal</th>
<th>diagonal coefficients of the covariance matrix.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>permutation</td>
<td>vector of permutations done during the Cholesky decomposition.</td>
</tr>
<tr>
<td>in</td>
<td>parameter</td>
<td>vector of parameters. The vector may either be empty or contain two clipping parameters ((a, b)). With the clipping parameters, for a normal distribution, any random value lies in ([\mu - a\sigma, \mu + b\sigma]) where (\mu) is the mean of the random variable and (\sigma) is its standard deviation.</td>
</tr>
<tr>
<td>in</td>
<td>output</td>
<td>vector to be tested. This vector was generated using a covariance matrix with diagonal diagonal.</td>
</tr>
</tbody>
</table>

Returns

true if the vector satisfies the constraints.

Definition at line 349 of file TRNGPerturbationManager.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/TRNGPerturbationManager.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/TRNGPerturbationManager.cxx

48.51 Verdandi::UnscentedKalmanFilter Class Reference

This class implements the unscented Kalman filter.

```
#include <UnscentedKalmanFilter.hxx>
```

Inheritance diagram for Verdandi::UnscentedKalmanFilter:

Verdandi::UnscentedKalmanFilter

Verdandi::VerdandiBase

Public Types

- typedef Model::state::value_type Ts
  
  Value type of the model state.
- typedef Model::state_error_variance_row model_state_error_variance_row
  
  Type of a row of the background error variance.
- typedef Model::state model_state
  
  Type of the model state vector.
- typedef Model::matrix_state_observation matrix_state_observation
  
  Type of the model/observation crossed matrix.
- typedef Model::state_error_variance model_state_error_variance
  
  Type of the background error variance.
- typedef Model::tangent_linear_operator model_tangent_linear_operator
Type of the tangent linear model.

- typedef ObservationManager::observation::value_type To
  Value type of the observation vector.

- typedef ObservationManager::tangent_linear_operator observation_tangent_linear_operator
  Type of the tangent linear observation operator.

- typedef ObservationManager::tangent_linear_operator_row observation_tangent_linear_operator_row
  Type of a row of the tangent linear observation operator.

- typedef ObservationManager::observation observation
  Type of the observation vector.

- typedef Vector<Ts, VectFull, MallocAlloc<Ts>> sigma_point
  Type of the sigma point vector.

- typedef Vector<sigma_point, Collection> sigma_point_collection
  Type of the sigma point collection.

- typedef Matrix<Ts, General, RowMajor, MallocAlloc<Ts>> sigma_point_matrix
  Type of the sigma point matrix.

- typedef Vector<model_state, Collection> state_collection
  Type of the state vector collection.

- typedef Vector<observation, Collection> observation_collection
  Type of the observation vector collection.

Public Member Functions

- UnscentedKalmanFilter()
  Main constructor.

- ~UnscentedKalmanFilter()
  Destructor.

- void Initialize(string configuration_file, bool initialize_model=true, bool initialize_observation_manager=true)
  Initializes the driver.

- void Initialize(VerdandiOps &configuration, bool initialize_model=true, bool initialize_observation_manager=true)
  Initializes the driver.

- void InitializeStep()
  Initializes a step for the unscented Kalman filter.

- void Forward()
  Performs a step forward with assimilation.

- void Prediction()
  Performs a forecast step.

- void Analyze()
  Computes an analysis.

- void FinalizeStep()
  Finalizes a step for the model.

- void Finalize()
  Finalizes the model.

- bool HasFinished()
  Checks whether the model has finished.

- Model & GetModel()
  Returns the model.

- ObservationManager & GetObservationManager()
  Returns the observation manager.

- OutputSaver & GetOutputSaver()
Returns the output saver.
- string GetName () const
  Returns the name of the class.
- void Message (string message)
  Receives and handles a message.

Static Public Member Functions

- static void StaticMessage (void ∗object, string message)
  Receives and handles a message with a static method.

Protected Attributes

- Model model_
  Underlying model.
- ObservationManager observation_manager_
  Observation manager.
- model_state_error_variance background_error_variance_
  Background error covariance matrix (B).
- int iteration_
  Iteration.
- string configuration_file_
  Path to the configuration file.
- string model_configuration_file_
  Path to the model configuration file.
- string observation_configuration_file_
  Path to the configuration file for the observation manager.
- map< string, bool > option_display_
  Display options.
- int Nstate_
  Dimension of the state.
- int Nobservation_
  Number of observations.
- bool analyze_first_step_
  Should an analysis be computed at the first step?
- string blue_computation_
  Computation mode for BLUE: "vector" or "matrix".
- string covariance_computation_
  Computation mode for covariance: "vector" or "matrix".
- string sigma_point_type_
  Choice of sigma-points.
- sigma_point_collection sigma_point_collection_
  Collection of sigma-points.
- sigma_point alpha_i_
  Coefficient vector associated with sigma-points.
- sigma_point_matrix X_i_trans_
  Transpose of \[X_n^\top\] \(\{\}\).
- bool alpha_constant_
  Boolean to indicate if the coefficients alpha are constants.
- Ts alpha_
48.51.1 Detailed Description

This class implements the unscented Kalman filter.

48.51.2 Constructor & Destructor Documentation

48.51.2.1 Verdandi::UnscentedKalmanFilter::UnscentedKalmanFilter ( )

Main constructor.
Builds the driver and reads option keys in the configuration file.

Parameters

| in       | configuration_file | configuration file |

Definition at line 50 of file UnscentedKalmanFilter.cxx.

48.51.3 Member Function Documentation

48.51.3.1 void Verdandi::UnscentedKalmanFilter::Analyze ( )

Computes an analysis.
Whenever observations are available, it computes BLUE.
Definition at line 359 of file UnscentedKalmanFilter.cxx.

48.51.3.2 Model & Verdandi::UnscentedKalmanFilter::GetModel ( )

Returns the model.
Definition at line 592 of file UnscentedKalmanFilter.cxx.

48.51.3.3 string Verdandi::UnscentedKalmanFilter::GetName ( ) const [virtual]

Returns the name of the class.
Reimplemented from Verdandi::VerdandiBase.
Definition at line 629 of file UnscentedKalmanFilter.cxx.
48.51.3.4 ObservationManager & Verdandi::UnscentedKalmanFilter::GetObservationManager ( )

Returns the observation manager.

Returns

The observation manager.

Definition at line 605 of file UnscentedKalmanFilter.cxx.

48.51.3.5 OutputSaver & Verdandi::UnscentedKalmanFilter::GetOutputSaver ( )

Returns the output saver.

Returns

The output saver.

Definition at line 617 of file UnscentedKalmanFilter.cxx.

48.51.3.6 bool Verdandi::UnscentedKalmanFilter::HasFinished ( )

Checks whether the model has finished.

Returns

True if no more data assimilation is required, false otherwise.

Definition at line 580 of file UnscentedKalmanFilter.cxx.

48.51.3.7 void Verdandi::UnscentedKalmanFilter::Initialize ( string configuration_file, bool initialize_model = true, bool initialize_observation_manager = true )

Initializes the driver.

Initializes the model and the observation manager. Optionally computes the analysis of the first step.

Definition at line 83 of file UnscentedKalmanFilter.cxx.

48.51.3.8 void Verdandi::UnscentedKalmanFilter::Initialize ( VerdandiOps & configuration, bool initialize_model = true, bool initialize_observation_manager = true )

Initializes the driver.

Initializes the model and the observation manager. Optionally computes the analysis of the first step.

Definition at line 97 of file UnscentedKalmanFilter.cxx.

48.51.3.9 void Verdandi::UnscentedKalmanFilter::InitializeStep ( )

Initializes a step for the unscented Kalman filter.

Initializes a step for the model.

Definition at line 226 of file UnscentedKalmanFilter.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/UnscentedKalmanFilter.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/method/UnscentedKalmanFilter.cxx
48.52 Verdandi::Variable Class Reference

This class enables to define a variable for the output saver.

#include <Variable.hxx>

Public Member Functions

- **Variable ()**
  Default constructor.
- **Variable (string mode, string file, bool has_to_empty_file=false)**
  Main constructor.
- **Variable (const Variable &variable)**
  Copy constructor.
- **~Variable ()**
  Destructor.
- **void SetMode (string mode)**
  Mode accessor.
- **void SetFile (string file)**
  Filename accessor.
- **void HasToEmptyFile (bool has_to_empty_file)**
  Empty accessor.
- **string GetMode () const**
  Mode accessor.
- **string GetFile () const**
  File accessor.
- **bool HasToEmptyFile () const**
  Empty accessor.
- **void Display () const**
  Displays on screen the saving mode and the output filename.

48.52.1 Detailed Description

This class enables to define a variable for the output saver.

48.52.2 Constructor & Destructor Documentation

48.52.2.1 Verdandi::Variable::Variable ( string mode, string file, bool has_to_empty_file = false )

Main constructor.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>in</strong> mode</td>
<td>saving format (e.g., &quot;binary&quot;, &quot;text&quot;).</td>
</tr>
<tr>
<td><strong>in</strong> file</td>
<td>path to the output file.</td>
</tr>
<tr>
<td><strong>in</strong> has_to_empty_file</td>
<td>Boolean that indicates if the file has to be emptied.</td>
</tr>
</tbody>
</table>

Definition at line 52 of file Variable.cxx.
48.52.2.2 Verdandi::Variable::Variable ( const Variable & variable )

Copy constructor.

Parameters

| in  | variable | instance to be copied. |

Definition at line 62 of file Variable.cxx.

48.52.3 Member Function Documentation

48.52.3.1 string Verdandi::Variable::GetFile ( ) const

File accessor.

Returns the path to the output file.

Returns
  The path to the output file.

Definition at line 125 of file Variable.cxx.

48.52.3.2 string Verdandi::Variable::GetMode ( ) const

Mode accessor.

Returns the saving mode.

Returns
  The saving format.

Definition at line 115 of file Variable.cxx.

48.52.3.3 void Verdandi::Variable::HasToEmptyFile ( bool has_to_empty_file )

Empty accessor.

Sets the value of the boolean empty.

Parameters

| in   | has_to_empty_file | Boolean that indicates if the file has to be emptied. |

Definition at line 105 of file Variable.cxx.

48.52.3.4 bool Verdandi::Variable::HasToEmptyFile ( ) const

Empty accessor.

Indicates if the file has to be emptied.
Returns

Boolean that indicates if the file has to be emptied.

Definition at line 135 of file Variable.cxx.

48.52.3.5 void Verdandi::Variable::SetFile ( string file )

Filename accessor.
Sets the path to the output file.

Parameters

| in | file | path to the output file. |

Definition at line 94 of file Variable.cxx.

48.52.3.6 void Verdandi::Variable::SetMode ( string mode )

Mode accessor.
Sets the saving mode.

Parameters

| in | mode | saving format (e.g., "binary", "text"). |

Definition at line 84 of file Variable.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/share/Variable.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/share/Variable.cxx

48.53 Verdandi::VerdandiBase Class Reference

Base class for Verdandi objects.

#include <VerdandiBase.hxx>

Inheritance diagram for Verdandi::VerdandiBase:
Public Member Functions

- VerdandiBase ()
  Default constructor.
- virtual VerdandiBase ()
  Destructor.
- virtual string GetName () const
  Returns the name of the class.
- virtual void Message (string message)
  Receives and handles a message.

Static Public Member Functions

- static void StaticMessage (void *object, string message)
  Receives and handles a message with a static method.

48.53.1 Detailed Description

Base class for Verdandi objects.

Template Parameters

\[ T \]  the type of floating-point numbers.
48.53.2 Member Function Documentation

48.53.2.1 string Verdandi::VerdandiBase::GetName ( ) const  [virtual]

Returns the name of the class.

Returns

The name of the class.

Reimplemented in Verdandi::LinearObservationManager, Verdandi::PetscLinearObservationManager, Verdandi::GridToNetworkObservationManager, Verdandi::ShallowWater, Verdandi::ClampedBar, Verdandi::PetscClampedBar, Verdandi::LevelSetObservationManager, Verdandi::QuadraticModel, Verdandi::ReducedOrderUnscentedKalmanFilter, Verdandi::ReducedMinimax, Verdandi::PythonModel, Verdandi::ReducedOrderExtendedKalmanFilter, Verdandi::HamiltonJacobiBellman, Verdandi::UnscentedKalmanFilter, Verdandi::EnsembleKalmanFilter, Verdandi::FourDimensionalVariational, Verdandi::PythonObservationManager, Verdandi::OptimalInterpolation, Verdandi::ExtendedKalmanFilter, Verdandi::ExtendedMinimaxFilter, Verdandi::FrontPositionObserver, Verdandi::ModelTemplate, Verdandi::MonteCarlo, Verdandi::ObservationManagerTemplate, Verdandi::Lorenz, Verdandi::Nudging, Verdandi::ObservationGenerator, Verdandi::ForwardDriver, and Verdandi::CheckingModel.

Definition at line 51 of file VerdandiBase.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/share/VerdandiBase.hxx
- /nas/home2/n/nclaude/Travail/code/Verdandi/verdandi-1.7/share/VerdandiBase.cxx

48.54 Verdandi::VerdandiOps Class Reference

This class extends the Ops::Ops class with Verdandi-related features.

#include <VerdandiOps.hxx>

Public Member Functions

- VerdandiOps ()
  Default constructor.
- VerdandiOps (string file_path)
  Main constructor.
- ~VerdandiOps ()
  Destructor.
- template<class TD , class T > void Set (string name, string constraint, const TD &default_value, T &value)
- template<class T > void Set (string name, string constraint, T &value)
- template<class T > void Set (string name, T &value)
- template<class T > T Get (string name)
- template<class T > T Get (string name, string constraint)
- template<class T > T Get (string name, string constraint, const T &default_value)
- template<class T > bool Is (string name)
Protected Member Functions

- \texttt{template\textless\texttt{class T , class Allocator \textgreater\ \ void \ SetValue (string name, string constraint, const Seldon::Vector\textless{} T, VectFull, Allocator \textgreater{} \&default_value, bool with_default, Seldon::Vector\textless{} T, VectFull, Allocator \textgreater{} \&value)}
- \texttt{template\textless\texttt{class T , class Allocator \textgreater\ \ void \ SetValue (string name, string constraint, const Seldon::Vector\textless{} T, PETScPar, Allocator \textgreater{} \&default_value, bool with_default, Seldon::Vector\textless{} T, PETScPar, Allocator \textgreater{} \&value)}
- \texttt{template\textless\texttt{class T , class Allocator \textgreater\ \ void \ SetValue (string name, string constraint, const vector\textless{} Seldon::Vector\textless{} T, VectFull, Allocator \textgreater{} \&default_value, bool with_default, vector\textless{} Seldon::Vector\textless{} T, VectFull, Allocator \textgreater{} \&value)}
- \texttt{template\textless\texttt{class T , class Prop , class Storage , class Allocator \textgreater\ \ void \ SetValue (string name, string constraint, const Seldon::Matrix\textless{} T, Prop, Storage, Allocator \textgreater{} \&default_value, bool with_default, Seldon::Matrix\textless{} T, Prop, Storage, Allocator \textgreater{} \&value)}
- \texttt{template\textless\texttt{class T , class Prop , class Storage , class Allocator \textgreater\ \ void \ SetValue (string name, string constraint, const vector\textless{} Seldon::Matrix\textless{} T, Prop, Storage, Allocator \textgreater{} \&default_value, bool with_default, vector\textless{} Seldon::Matrix\textless{} T, Prop, Storage, Allocator \textgreater{} \&value)}
- \texttt{template\textless\texttt{class T , class Prop , class Storage , class Allocator \textgreater\ \ void \ SetValue (string name, string constraint, const vector\textless{} Seldon::Matrix\textless{} T, Prop, PETScMPIAIJ, Allocator \textgreater{} \&default_value, bool with_default, vector\textless{} Seldon::Matrix\textless{} T, Prop, PETScMPIAIJ, Allocator \textgreater{} \&value)}
- \texttt{template\textless\texttt{class T , class Prop , class Storage , class Allocator \textgreater\ \ void \ SetValue (string name, string constraint, const vector\textless{} Seldon::Matrix\textless{} T, Prop, PETScMPIAIJ, - Allocator \textgreater{} \&default_value, bool with_default, vector\textless{} Seldon::Matrix\textless{} T, Prop, PETScMPIAIJ, Allocator \textgreater{} \&value)}
- \texttt{template\textless\texttt{class T , class Prop , class Storage , class Allocator \textgreater\ \ void \ SetValue (string name, string constraint, const vector\textless{} Seldon::Matrix\textless{} T, Prop, Storage, Allocator \textgreater{} \&default_value, bool with_default, vector\textless{} Seldon::Matrix\textless{} T, Prop, Storage, Allocator \textgreater{} \&value)}
- \texttt{template\textless\texttt{class T , class Prop , class Storage , class Allocator \textgreater\ \ bool \ IsParam (string name, Seldon::Vector\textless{} T, VectFull, Allocator \textgreater{} \&value)}
- \texttt{template\textless\texttt{class T , class Prop , class Storage , class Allocator \textgreater\ \ bool \ IsParam (string name, Seldon::Matrix\textless{} T, Prop, Storage, Allocator \textgreater{} \&value)}
- \texttt{bool \ CheckConstraint (string expression)}

48.54.1 Detailed Description

This class extends the Ops::Ops class with Verdandi-related features.

48.54.2 Constructor & Destructor Documentation

48.54.2.1 Verdandi::VerdandiOps::VerdandiOps ( )

Default constructor.
Nothing is performed. The Lua state is set to NULL.
Definition at line 43 of file VerdandiOps.cxx.

48.54.2.2 Verdandi::VerdandiOps::VerdandiOps ( string file_path )

Main constructor.
The Lua configuration file is loaded and run. An exception may be raised during this evaluation.
Parameters

| in | file_path | path to the configuration file. |

Definition at line 53 of file VerdandiOps.cxx.
48.54.2.3 Verdandi::VerdandiOps::~VerdandiOps()

Destructor.
Destroys the Lua state object.
Definition at line 61 of file VerdandiOps.cxx.

The documentation for this class was generated from the following files:

- /nas/home2/nclaudio/Travail/code/Verdandi/verdandi-1.7/share/VerdandiOps.hxx
- /nas/home2/nclaudio/Travail/code/Verdandi/verdandi-1.7/share/VerdandiOps.cxx


