Verdandi: a Generic Data Assimilation Library

Claire Mouton, Vivien Mallet
In collaboration with Dominique Chapelle,
Philippe Moireau and Marc Fragu

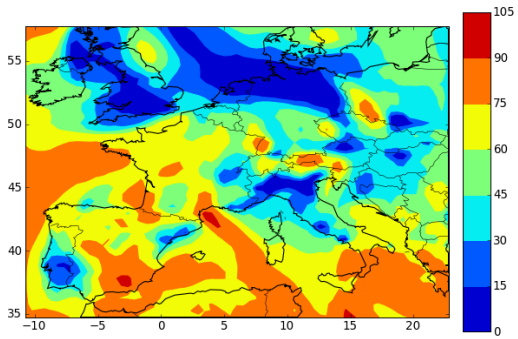
Second ADAMS Meeting
October 27^{th} 2009

1. Scientific context
   - Generality of data assimilation
   - What is the library designed for ?
2. Technical context
   - Taking advantage of modern tools
   - Verdandi languages
   - The dependences
3. Verdandi
   - Design
   - Current Verdandi version
   - Demonstration via the high-level interface
4. Perspectives
   - Data assimilation methods in Verdandi
   - Tools in Verdandi
   - Other perspectives

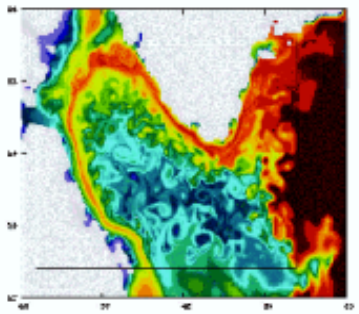
Generality of data assimilation

Data assimilation examples

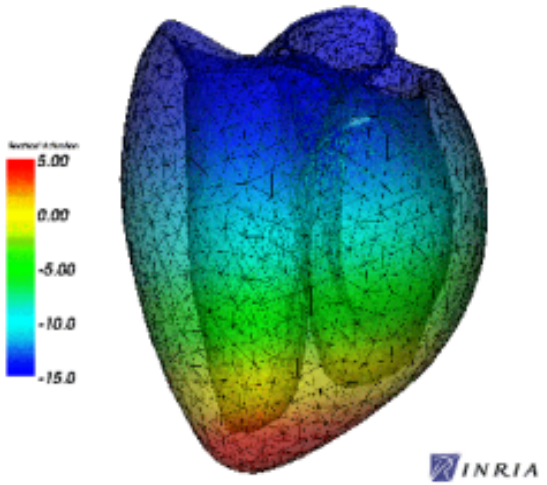
Air quality : ozone concentration map (CLIME team).



Oceanography : temperature field map (MOISE team).



Heart simulation : electrical activation during heartbeats (MACS team).



Generality of data assimilation methods

$$\mathbf{x}^f_{n+1} = \mathrm{M}_{n \to n+1}(\mathbf{x}^a_n)$$
$$\mathbf{x}^a_n = \mathbf{x}^f_n + \mathbf{K}(\mathbf{y}_n - \mathbf{H}[\mathbf{x}^f_n])$$

The methods are written independently of the system and can be applied to a great number of systems.

$\Rightarrow$ The methods are generic and can be put together in a library.

For example : optimal interpolation, Kalman filters, 4DVar.

What is the library designed for ?

- To make easier the application of methods to a great number of problems.
- To make perennial and put in common the developments.
- To improve the broadcast of data assimilation work inside INRIA (between teams) and with research partners.

- The users :
  Specialists (modular framework to work and exchange),
  Entrants (methods directly usable).
  The users provide the numerical model and the observations with the appropriate interface.

Taking advantage of modern tools

- **Long term design, accessibility, extensibility**
  Ensured by modern tools, an appropriate design and the respect of a coding standard.
- **Open development thanks to a free LGPL license**
  Anyone can have a copy of the library code, develop locally any feature and send his/her contribution to a shared reference code deposit (managed by Git). Same organization for the documentation.
- **Easy installation**
  Compilation with SCons.
- **Reliability**
  Unitary tests (CppUnit).

Verdandi languages

- **The core of the library**
  C++ ensures modularity, efficiency and provides many useful resources.

- **Higher-level interface**
  Allows to work with objects and methods without significant loss of performance (see the following demonstration).
  Automatically built interface to Python (high-level language providing free resources for scientific computing).

The dependences

- **Linear algebra library**
  Seldon chosen among 27 libraries studied (3 of them tested) thanks to its extend of the interface to Blas and Lapack, the availability of sparse matrices and vectors, the performance, the portability and the local mastering (`http://seldon.sourceforge.net/`).

- **Configuration file parser library**
  GetPot chosen among 26 libraries studied (2 of them tested) (`http://getpot.sourceforge.net/`).

What is an object ?

A C++ concept : a box containing data and functions.

The model can be seen as an object, providing an interface to work with it. The model itself can be written in Fortran, C or C++.

The observations are managed through an object called observation manager.

The data assimilation is driven via an object called data assimilation driver.

All these objects can interact through their interfaces.

Design

- Simulation without data assimilation :

```
model.Initialize();
while (!model.HasFinished())
    model.Forward();
```

- Simulation with sequential data assimilation :

```
model.Initialize();
observation_manager.Initialize(model);
while (!model.HasFinished())
{
  model.Forward();
  observation_manager.SetDate(model.GetDate());
  if (observation_manager.HasObservation())
      ...  model.GetState() += K * innovation;
}
```

Synthesis : three main object types

- Data assimilation methods :

  The model and the observation manager are part of this object.

  ```
  Initialize(configuration_file)
  Analyze()
  ```

- Models :

  ```
  Initialize(configuration_file)
  Forward()
  HasFinished()
  GetBackgroundErrorCovariance(i, j)
  ```

- Observation managers :

  ```
  Initialize(configuration_file)
  LoadObservation()
  ApplyOperator(x)
  GetObservationErrorCovariance(i, j)
  ```

Current Verdandi version

- Data assimilation method : optimal interpolation.
- Observation manager : linear observation manager, including the case of a diagonal operator and the case of a model grid to an observation network.
- Models : “shallow-water” (oceanography) and “clamped-bar” (mechanics).
- Management of the case of sparse matrices (error variance and observation operator matrices).

Demonstration via the high-level interface

```
host<~/> ipython -pylab
In [1]: method = verdandi.AssimilationMethod(''configuration_file.cfg'')
In [2]: method.Initialize(''configuration_file.cfg'')
In [3]: model = method.GetModel()
In [4]: model.GetCurrentDate()
Out[4]: 0
```

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

```
In [5]: method.Forward()
In [6]: model.GetCurrentDate()
Out[6]: 1
In [7]: model.Forward()
In [8]: model.GetCurrentDate()
Out[8]: 2

In [9]: for i in range(1008):
....:         model.Forward()
In [10]: model.GetCurrentDate()
Out[10]: 1010
```

Demonstration via the high-level interface

Computes the analysis and checks its effect by printing the state vector before and after the analysis :

```
In [11]: state_vector = seldon.VectorDouble()

In [12]: model.GetState(state_vector)
In [13]: state_vector.Print()
1.01678819980058454 68 1.0150381428637218484 ...

In [14]: method.Analyze()
In [15]: model.GetState(state_vector)
In [16]: state_vector.Print()
1.0167882075200211922 1.0150381458769643928 ...
```

Sets the state to an arbitrary value :

```
In [17]: state_vector[0] = 0
In [18]: model.SetState(state_vector)
In [19]: model.GetState(state_vector)
In [20]: state_vector.Print()
0  1.0150381458769643928 ...
```

Perspectives : data assimilation methods in Verdandi

- Basic method :
  nudging, optimal interpolation.
- Kalman filters :
  Kalman filter, ensemble Kalman filter, low-rank Kalman filters, the reduced-rank square root (RRSQRT) Kalman filter, mixed ensemble and RRSQRT Kalman filter, singular evolutive extended Kalman (SEEK), singular evolutive interpolated Kalman (SEIK), unscented Kalman filter.
- Variational methods :
  3D-Var, 4D-Var, incremental 4D-Var ?
- Ensemble forecast with sequential aggregation :
  least-squares methods (including ridge regression and Lasso), machine learning methods (gradient descent, exponentiated gradient, ...), statistical methods (dynamic linear regression, ...).
- At least one particle filter.
- Parameter estimation whenever relevant.

Perspectives : tools in Verdandi

- Models :
  - Test models : heart simulation (MACS), oceanographic (MOISE), air quality (Polyphemus), image processing (Fluminance).
  - Generic tangent linear model based on finite differences, validation of tangent linear models and adjoint models.
- Observation managers :
  - Template observation operator.
  - Observation operator mapping from gridded data to pointwise data (observations taken at a list of locations, like in a network), at least for 1D, 2D and 3D grids.
  - Generation of synthetic observations (at a list of locations, for instance) based on a model, possibly with perturbations
- Error statistics :
  - Classical error covariance matrices (diagonal, Gaussian form, Balgovind form).
  - χ^2 analysis.

Perspectives

- Manage the case of distributed vectors and matrices.
- Parallel computing.
- Provide documentations for developers (Doxygen) and users (generic example, code to complete).

Thank you for your attention.