Variational Data Assimilation & Model Learning

Jerome Monnier

INSA Toulouse

Department of Applied Mathematics

Variational Data Assimilation & Model Learning

Assimilating datasets into physically-based models, calibrating models, identifying model parameters, learning model terms

by Jérôme Monnier

Fall 2020

Keywords: inverse problems, model-based feedback control, PDE models, optimisation, gradient-based algorithms, adjoint method, best estimate, error covariances, Earth sciences.

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

INSTITUT DE MATHÉMATIQUES de TOULOUSE

Université de Toulouse

Contents

1

[1]The sections indicated with a * are "to go further sections". These sections can be skipped as a first reading, or if you are not interested in deeper mathematical basis, mathematical proofs.

WhereTo & How

Language
At INSA Toulouse, this course is tough in GlobeEngliche excepted if *all* attendants understand French sufficiently well.

Goal of this course
- To formulate the Variation Data Assimilation (VDA) equations aiming at fusing a physical-based model, databases and prior information (e.g. covariance operators).
- To design optimal control algorithms for systems governed by a PDE or an ODE (or even SDE).
- To compute gradients by deriving the adjoint equations.
- To formulate real-like numerical modelling problems by combining "optimally": mathematical - physical equations (PDE models mainly) and databases containing measurements of the phenomena.
- To perform model calibration, parameter identification, local sensitivity analysis by assimilating the available data into the model.
- To identify (learn) model terms from datasets.
- To understand the link between the VDA approach, the Bayesian approach and the sequential filters approaches (Kalman's filter).
- To known to assess computational codes including the adjoint code and the resulting gradient.
- To generate an adjoint code by employing "Automatic Differentiation" (source-to-source code transformation such as Tapenade tool) (also called "backward propagation gradient" in machine learning).

At the end, the students are supposed to be able :
- To set up a data assimilation inverse formulation e.g. in (geophysical) fluid mechanics, structural mechanics, biology etc in presence of databases, measurements.
- To compute efficiently the gradient with a large number of parameters by deriving the adjoint model.
- To design the complete optimisation process,
- To perform local sensitivities analysis, to calibrate the model, to estimate uncertain parameters by assimilating the data into the physical-based model (PDE but also potentially SDE).
- To learn model terms (ODE or PDE) from datasets.

Content

Fundamentals: inverse problems, least-squares solutions (linear and not), SVD, optimisation, regularisation of optimisation problems.

Examples of inverse problems, parameter identification, model calibration, data assimilation for prediction.

Basics of optimal control:
- ODE systems: the Linear-Quadratic (LQ) case, Pontryagin's principle, the Hamiltonian.
- Non-linear PDE systems (steady and unsteady): gradient computation, adjoint equations, optimality system, the Lagrangian.
- Variational Data Assimilation (VDA) formulation: cost function gradient, gradient-based optimisation, regularisation terms, connections with prior probabilistic model - covariances operators, connection with the Kalman's filter and the Bayesian view, 4D-var algorithm(s).

Examples and exercises.

A programming practical: assimilation of altimetry type data into a river flow model (non-linear advection-diffusion model).
Implementation in Python (or FreeFem++).

Algorithmic Differentiation, Automatic differentiation (source-to-source e.g. with Tapenade software); link with the "backward propagation gradient".

*The sections indicated with a * are "to go further sections". These sections can be skipped as a first reading, or if you are not interested in deeper mathematical basis, mathematical proofs.*

Keywords PDE models, uncertainties, data basis, variational calculus, optimal control, adjoint equations, variational data assimilation, sensitivities, prior probabilistic information, covariance operators, algorithmic differentiation.

Background required Differential calculus, PDE models, numerical analysis, optimisation, programming, basics of probability.
(Basics knowledge of neural networks - deep learning enables to make various links between these two complimentary approaches).

Duration At INSA Toulouse, this course is scheduled within 9 weeks, 25H of plenary classes, exercise sessions and a couple of sessions to complete the programming practical. The expected home work duration is 25H too.

Examinations A few $\approx$ 45mn long exams : on-line exercises with multiple-choice questionnaires .
1 programming practical work with a (short) written report + the source code.

Chapter 1

Fundamentals and practical

Data Assimilation (DA) aims at fusing "at best" the mathematical model and data - measurements (also called observations).
The latter may be in-situ or remote sensed (e.g. satellite or drones images); they can be big databases. The mathematical models are generally Partial Derivatives Equation (PDE) systems; (it may be a Stochastic Differential Equation (SDE) too, not addressed here). DA are physically-based processes solving inverse problems, inference problems.

Variational Data Assimilation (VDA) approach relies on an optimization formulation.
Common features with Artificial Neural Networks (ANN) employed for regression problems can be done; not really in the way the methods work but on the goals and the obtained results.

Note that mixing physically-based method and data-driven ones would lead to hybrid methods; this goal is a new and active research field, therefore out of the scope of the present course.
The present course will mainly focus on VDA and Model Learning.

The outline of this chapter is as follows.

Contents

1.1 Direct vs Inverse modeling

1.1.1 Direct, inverse

In the common sense, the term "model" denotes a *direct* model (also called *"forward"* model).

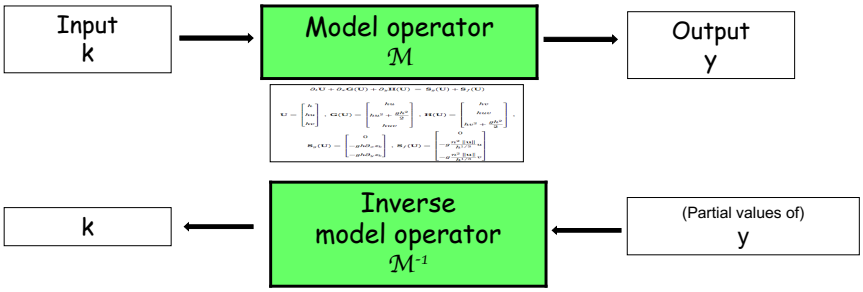


Figure 1.1: Representation of a direct model with its input "parameter" k (a-priori vectorial) and its output variable y (its "response"); and its inverse counterpart.

A direct problem based on the model operator $\mathcal{M}(.)$ consists to find (compute) a solution y given the input parameter k: find y, $y = \mathcal{M}(k)$.

The inverse problem consists to find (compute) k, given y i.e. solve: $k = \mathcal{M}^{-1}(y)$.

1.1.2 Examples

Example 1) (Parameter identification).
Find a polynomial $p(x)$ of degree n, of coefficients $(k_0, \cdots, k_n)$ that fit given values $(z_1, , ..., z_n)$ at given points $(x_1, ..., x_n)$.
The Lagrange interpolation problem is the inverse problem of the following direct problem: calculate the given polynomial at given points $(x_1, ..., x_n)$.

This example is a basic data assimilation problem, aiming at *identifying parameters* of the "model" $p(k; x)$.

Example 2) (Model term identification) (from [26]).
Given a real symmetric $n \times n$ matrix M and n real numbers $(\lambda_1, \cdots, \lambda_n)$, find a diagonal matrix D such that $(M + D)$ has the eigenvalues $(\lambda_1, \cdots, \lambda_n)$.
This problem is inverse to the following direct problem: compute the eigenvalues of the given matrix $(M + D)$.

This inverse problem aims at *identifying model terms*.

Example 3) (Model Learning).
Given the first few terms $(x_1, \cdots, x_m)$ of a sequence, find the law of formation of the sequence; that is find x_n for all n.
The corresponding direct problem is to evaluate the sequence $(x_n)_n$ given the law of formation (the model).

From [26]: this inverse problem is used on intelligence tests. In a mathematical point of view, such inverse problems have many solutions (that is why their use on intelligence tests has seemingly been criticized).

This example may be perceived as a simple *model learning* example.

Example 4) (PDE based DA problem).
Let us consider a diffusion phenomena in a material. The non homogeneous diffusivity (it may be a conductivity) of the material (eg. biological tissue) is denoted by $\nu(x)$. The model is the following.
Given $\nu(x)$ in the domain Ω, find the scalar quantity $u(x, t)$ (e.g. an electrical field or wave intensity) satisfying:

$$\begin{cases} \partial_t u(x,t) - div(\nu(x)\nabla u(x,t)) = 0 & \text{in } \Omega\times]0,T[\\ u(x,0) \;=\; u_0(x) & \text{in } \Omega \\ u(x,t) = u_d(x,t) & \text{in } \partial\Omega\times]0,T[\end{cases} \tag{1.1}$$

The initial condition u_0 and the value u_d at boundary are given.
The direct problem consists to solve this classical Boundary Value Problem.

The inverse problem is as follows.

One acquire *measurements* of the field $u(x,t)$ and the flux $(\nu(x)\partial_n u(x,t)$ on the boundary $\partial\Omega$.

Given these boundary measurements, one seek to determine the unknown-uncertain diffusivity coefficient $\nu(x)$ in the domain Ω (eg. a conductivity).

This inverse problem is the "impedance tomography" problem.
In the context of Electrical Impedance Tomography (EIT), the inverse problem aims at recovering conductivity (and permittivity) inside a body from surface measurements of electrical currents and potentials.
It is a potentially ill-posed problem (depending on the assumptions), see e.g. [L. Borcea, Inv. Problems (2002)].
EIT problem is still active research problem; it still posed challenging questions for mathematicians, numerician and experimentalists. "EIT is a noninvasive type of medical imaging in which the electrical conductivity, permittivity, and impedance of a part of the body is inferred from surface electrode measurements and used to form a tomographic image of that part" (from Wikipedia page).
See eg. Fig. 1.2.

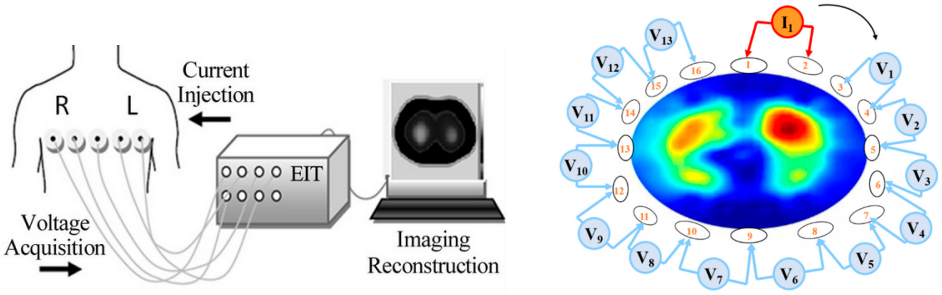


Figure 1.2: Electrical Impedance Tomography (EIT) for Cardio-Pulmonary Monitoring: voltage measurements around the thorax using an EIT system with 16 electrodes. (R) Image extracted from C. Putensen et al., J. Clinical Medecine (2019).

The reader may consult e.g. [26] or [3] to read other standard instructive inverse problems in various technological contexts; the mathematical problems involves integral equations, PDE systems etc.

1.1.3 Ill-posedness, well-posedness

In the Hadamard sense[1], a model is well-posed if the following conditions hold:
i) it admits an unique solution,
ii) the solution depends continuously on the data or input parameters.

The first condition i) is entirely related to the space the solution is sough in.
The second condition ii) is a stability condition. This property is crucial too.
Indeed, if this condition is not satisfied then any measurement or numerical error generates large errors in the model response (i.e. a highly perturbed solution).

In all the sequel, it will be assumed that the direct model is well-posed; this is a necessary condition to go further !...

This assumption implies in particular that the "model operator"

$$\boxed{\mathcal{M} : k \in \mathcal{K} \mapsto y \in \mathcal{Y} \text{ is continuous.}} \tag{1.2}$$

Note that thanks to the *open mapping theorem*, see e.g. [11]:
If $\mathcal{M}$ is linear and continuous with $\mathcal{X}$ and $\mathcal{Y}$ Banach spaces, then the inverse model operator $\mathcal{M}^{-1}$ is continuous

Exercise. Propose a general PDE-based example which is trivially well-posed.
(*Hint : mind to the linear elliptic BVP, coercitive in a Hilbert space V, with the Lax-Milgram theory applying).* □

In real-life modeling, even if the direct problem is well-posed, the inverse problem is very often severely ill-posed !...

Ill-posed inverse problems are somehow the opposite of well-posed direct problems.
Direct problems are usually the way that can be solved easily (compared to the inverse problem). Actually, direct and inverse models are back-and-forth transmission of information between k and y. Roughly, "direct problem involves going from cause to effect, whereas the inverse problem attempts to go from the effects to the cause" (from [2]).

Many inverse problems in science and engineering consist to determine properties of some unmeasurable quantities (in space and/or in time).
The information, data are far to be complete or even accurate. This lack of observations, data (both in terms of quantity and quality) is one of the source of the encountered difficulties to solve the inverse problems.
The measurements of Y are almost always incomplete (and inaccurate). Also, the direct models are often non-linear.

In real-life modeling, inverse problems are as common as direct problems. The inverse problem classes and techniques are extremely wide. Inverse modeling is a fast growing topic, requiring both stochastic and deterministic skills.

The historical and common applications fields are in geosciences (e.g. weather forecast, oil reservoirs, hydrology), neutronic (nuclear power plants), inverse scattering (e.g. seismology) and imaging (e.g. medical imaging, tomography, image restoration).

Data Assimilation (DA) problems are particular class of inverse problems. This course will focus in particular on *Variational Data Assimilation* (VDA) based on calculus variations and optimal control.

VDA solutions are somehow *least-square* solutions with a *physically-based model constraint.*

Then, let us recall basic properties of least-square solutions and Singular Value Decomposition (SVD) in the case of simple inverse problems.

1.2 Least-square solutions of regression problems

1.2.1 Linear least-square problems

Let assume that we have m measurements $(z_i)_{1\leq i\leq m}$ we seek to describe by a "model".
To do so, we choose to consider the following linear model with n unknown parameters $(k_i)_{1\leq i\leq n}$:

$$\left\{\begin{array}{rcl} a_{11}k_1 + ... + a_{1n}k_n &=& z_1 \\ ... &=& ... \\ a_{m1}k_1 + ... + a_{mn}k_n &=& z_m \end{array}\right.$$

We denote: $A = (a_{ij})_{1\leq i\leq m, 1\leq j\leq n}$ the chosen linear transformation (the linear model is given), A is a $m \times n$-matrix, $z \in \mathbb{R}^m$ the observation vector and $k \in \mathbb{R}^n$ the unknown input parameter vector.
This is a *identification parameter problem.* This problem reads as:

$$\text{Given } A, \text{ find } k \in \mathbb{R}^n \text{ such that: } Ak = z.$$

In the case there is as much parameters k_i as observations z_k (yes, that sounds weird...), i.e. $n = m$, the model is well-posed if and only if A is a full-rank matrix. In this case, it exists a unique set of parameters k describing exactly the data.

Still in the case $n = m$ but if the model is ill-posed in the sense $\text{rank}(A) < n$, then it exists solutions but they are not unique.
In this case, the kernel of A, $Ker(A)$, contains non zero vectors y such that: $Ay = 0$. If k^* is a solution then $(k^* + y)$ with $y \in Ker(A)$ is also solution.
In practice there is no reason to have $n = m$!...

In the case $n > m$ i.e. in the unlikely case there is more input parameters than observations, the system is under-determined. Generally, it exists solutions but they are non-unique.

In the case $n < m$ i.e. in the usual case there is less input parameters than observations, the system is over-determined. A-priori it does not exist any solution fitting all data.
Indeed, with m input parameters, it can exist a unique solution making fit the observations; but what about the extra $(n - m)$ "constraint equations" ?

Least-square solutions. Instead of seeking a solution satisfying each equation above (i.e. a solution making fit exactly all the observations), it is interesting to find a solution satisfying "at best" the system; in other words, a solution k minimizing the norm $\|Ak - z\|$.
Of course the choice of different norms will lead to different solutions...

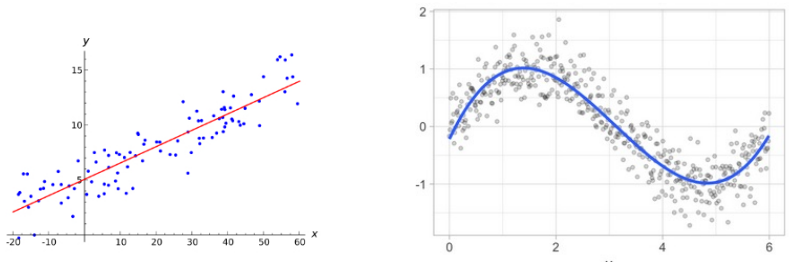


Figure 1.3: Linear least-square problem (with dense observations-measurements !). (L) The most simple case: linear regression; two parameters to identify. (R) An other simple case (polynomial, degree 3).

An easy choice of norm is the Euclidian norm $\|.\|_2$ since it is associated to a *scalar product* on contrary to the norms $\|.\|_1$ and $\|.\|_\infty$ for example.
Then the problem becomes: find $k^* \in \mathbb{R}^n$ such that

$$\boxed{j(k^*) = \min_{\mathbb{R}^n} j(k) \text{ with } j(k) = \frac{1}{2}\|Ak - z\|^2_{2,m}}$$

It is an unconstrained optimization problem in a convex set.
The functional j reads: $j(k) = \frac{1}{2}(A^TAk, k) - (Ak, z) + \frac{1}{2}(z, z)$. j is quadratic, convex since A^TA is positive, hence j admits a minimum in $\mathbb{R}^n$.
Furthermore, if A^TA is definite (it means $n \leq m$ and A is full rank, $\dim(\text{Im}(A) = m)$ then the solution is unique.
The gradient of j reads: $\nabla j(k) = A^TAk - A^Tz$. Then, the solution k has to satisfy the *necessary optimal condition*:

$$\boxed{A^TAk = A^Tz}$$

It is the *normal equations*.

Full rank case. In the case A full rank (i.e. $\dim(\text{Im}(A)) = m$), then the solution is unique since A^TA is symmetric positive definite. Even if A sparse, then A^TA is a-priori non sparse; also $K_2(A^TA) = (K_2(A))^2$, hence the normal equations can be an ill-conditioned system. Then, a good algorithm to solve the normal equations is a-priori not the Cholesky algorithm, especially if m large. Remembering that the 2-norm is preserved under orthogonal transformations, a better option is to solve the following equivalent system:

$$\min_{k \in \mathbb{R}^n} \frac{1}{2}\|QAk - Qz\|^2_{2,m}$$

with Q an orthogonal matrix.
By performing QR-factorizations (Householder's method), the resulting linear system to be inverted is a triangular $n \times n$ system, with its original conditioning number $K_2(A)$ preserved.

1.2.2 Ill-posed inverse problem & Tikhonov regularization

It has been mentioned above that in the case A is full rank (dim(Im(A)) = m) then the least-square solution (the solution of the normal equations) is unique. However, even if A is full rank, the system

may be ill-conditioned e.g. the smallest eigenvalue is extremely small. This is a frequent case in real-world problems !
In order to better define the problem or to select a solution with desirable properties, a "regularization term" is introduced.

The most classical way to regularize the inverse problem consists to compute $k^* \in \mathbb{R}^n$ such that $j(k^*) = \min_{\mathbb{R}^n} j(k)$ with

$$\boxed{j(k) = \frac{1}{2}\|Ak - z\|^2_{2,m} + \alpha_{reg}\frac{1}{2}\|Ck\|^2_{2,n}} \tag{1.3}$$

with C the regularization matrix and α_{reg} a positive scalar (the weight) making the balance between the two terms.
The added regularization term is a convex, differentiable term. It is the so-called *Tikhonov regularization method.*
(A. Tikhonov, Russian mathematician, 1906-1993). This method seems to be due to Phillips and Miller too.

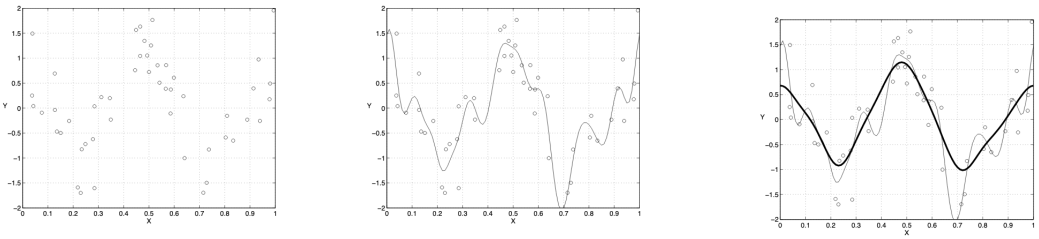


Figure 1.4: A fitting problem with/without Tikhonov regularization term. (L) The dataset to fit "at best" (= in the LS sense). (M) A LS optimal fitting curve (without regularization). (R) The same class of LS solution but with a regularization term. (Image extracted from a MIT course).

The solution of the corresponding normal equations reads:

$$\boxed{k^* = (A^T A + \alpha^2_{reg} C^T C)^{-1} \cdot A^T z}$$

The choice $C = Id$ provides the least-square solution with a minimal norm.
In real-world modeling problem, the regularization operator C may be chosen as a differential operator (eg. the gradient or Laplace operator) or a Fourier operator (e.g. filtering high frequencies).

Indeed, it turns out that numerous real-world direct models (phenomena) have the effect of low-pass filters.
As a consequence, the inverse operator acts as a high-pass filter since the eigenvalues (or singular values) are the largest in the reverse mapping when they are the smallest in the forward mapping.

Amplifying high frequency is not a desirable feature since it amplifies noise and uncertainties (e.g. of data). As a consequence, regularization operators play a crucial role in inverse formulations.

The weight parameter value α_{reg} is empirically set since its 'optimal' value is a-priori not known. Various methods are proposed to select a value; let us cite the discrepancy principle (Morozov's principle), cross-validation techniques and the L-curve method. The latter is briefly presented in next paragraph.

In statistics, the *Tikhonov regularization* is called the *ridge regression* method.
In Machine Learning (ML), it corresponds to the so-called *weight decay* method.

Other regularization terms and norms Other regularization terms can be considered; typically the 1-norm. In this case, the regularization term reads: $\alpha_{reg}\frac{1}{2}\|Cx\|_{1,n}$.
This term is still convex but it is non differentiable therefore leading to a non-differential convex optimization problem.
Obviously this is no longer a least square problem; this is the so-called *LASSO* problem.
The 1-norm is highly interesting for its *compress sensing* features.

On multi-objective optimization The introduction of regularization term naturally leads to a bi-objective optimization, see (1.3). As already mentioned, the weight parameter value α_{reg} has to be a-priori set. Obviously the computed optimal solution depends on α_{reg}.
No magic criteria nor method exist to determine an optimal value of α_{reg}. This weight coefficient has to respect a "good" trade-off between the two terms $j_{misfit}(\cdot)$ and $j_{reg}(\cdot)$.
An expertise of the modeled system is likely the best way to determine an optimal value of α_{reg}.
The so-called L-curve helps to visualize this trade-off, see Fig. 1.5.

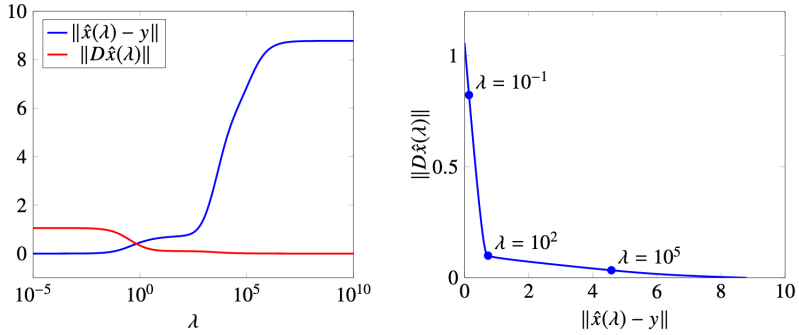


Figure 1.5: Bi-objective optimization: an example. The optimal computed optimal solution depends on the weight parameter α_{reg} (here denoted by λ). (L) The obtained minimal value of each cost function terms $j_{misfit}(k^*)$ and $j_{reg}(k^*)$ (here equal to $\|Dk\|$) vs α_{reg}. (R) The L-curve: values of $j_{reg}(k^*)$ vs values of $j_{reg}(k^*)$ for different α_{reg} values.

1.2.3 Non linear least-square problems

Let us consider the same problem as previously: set up a model describing "at best" m available data $z_i,\ 1 \leq i \leq m$ but using a *non-linear* model. In this case, the corresponding least-square formulation

reads:

$$\boxed{j(k^*) = \min_{\mathbb{R}^n} j(k) \text{ with } j(k) = \frac{1}{2}\|M(k)\|^2_{2,m}} \tag{1.4}$$

with M defined from $\mathbb{R}^n$ into $\mathbb{R}^m$, M *non linear* in k. Again, it is an unconstrained optimization problem in a convex set.
In the previous linear case, M was equal to: $M(k) = Ak - z$.

Note that neither the existence of a solution, nor its uniqueness is guaranteed without (strong) assumptions. Non-linear problems are much more difficult than linear ones. Bellow, and more generally in this course, one consider heuristic computational algorithms only.

Examples

Example 1) The location problem (eg by GPS or Galileo). The location problem by Global Navigation System Satellites (GNSS) consists to estimate x, $x \in \mathbb{R}^3$ (the location value).
The measurements are distances d_i^{obs} from given locations a_i with $i = 1, \cdots, m$ (the m satellites):

$$d_i^{obs} = \|a_i - x^{exact}\|_{2,\mathbb{R}^3} + \varepsilon_i \quad i = 1, .., m$$

where ε denotes a noise (the inacurracy of the measurements).

The least-square problem to be solved is:

$$\min_{x \in \mathbb{R}^3} j(x) \tag{1.5}$$

with: $j : \mathbb{R}^3 \to \mathbb{R}$,

$$j(k) = \|\|a_i - k\|_{2,\mathbb{R}^3} - d_i^{obs}\|^2_{2,\mathbb{R}^m} = \sum_{i=1}^{m} \left(\|a_i - k\|_{2,\mathbb{R}^3} - d_i^{obs}\right)^2$$

The functional $j(\cdot)$ is non convex, see Fig. 1.6.

Example 2) A standard fitting problem The goal is to fit at best a (dense) dataset; the regression problem plotted in Fig. 1.7(L). The chosen functional ("model") is: $j(k_1, \cdots, k_4) = k_1 \exp(k_2 x) \cos(k_3 x + k_4)$ with the four parameters k_i to identify.

Example 3) Classification problem A basic way to solve a binary classification problem is to solve the non-linear least square problem defined by:

$$j(k_1, \cdots, k_p) = \sum_{i=1}^{m} \left(\Phi(k_1 f_1(x_i) + \cdots + k_p f_p(x_i)) - d_i\right)^2$$

where the functions $f_j(\cdot)$ are given. $\Phi(u)$ is he sigmoidal function; $\Phi(u) = \frac{\exp(u) - \exp(-u)}{\exp(u) + \exp(-u)}$. Φ is a differentiable function approximating the sign function, Fig. 1.7(R).

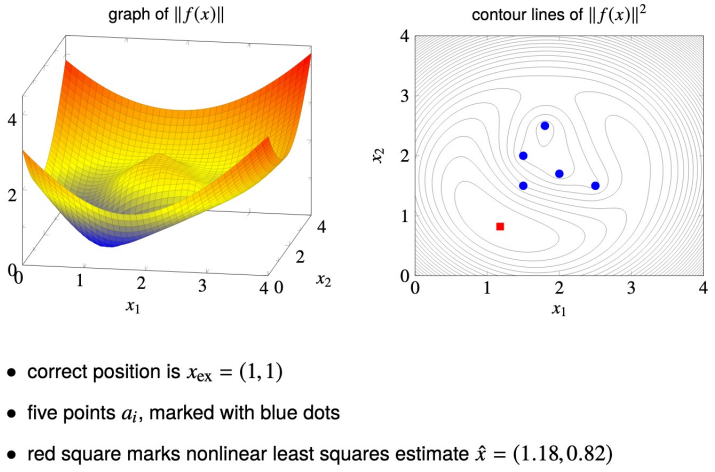


Figure 1.6: Non-linear least square problems: the location problem. (Image extracted from Vandenberghe's UCLA course). (L) The graph of $j(k)$, non convex functional in k. (R) Iso-values of $j(k)$, data, exact solution and the least-square one.

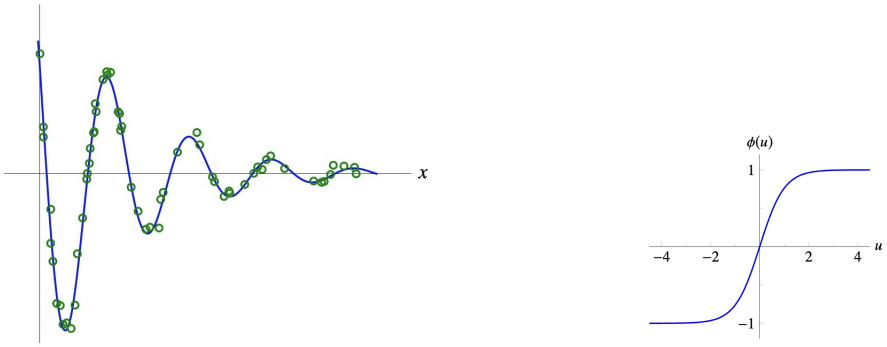


Figure 1.7: Non-linear least square problems: examples. (L) A fitting problem (with dense data); 4 parameters to identify. (R) The (differentiable) sigmoid function employed to solve binary classification problems.

Derivatives & optimality conditions

Let us write the necessary first order optimality condition for the general non-linear square problem (1.4). This optimality condition reads:

$$\nabla j(k) = 0$$

We have: $j(k) = \frac{1}{2} M^T(k) M(k)$. Let us denote the Jacobian of M as follows:

$$DM(k) = \left(\frac{\partial M_i}{\partial k_j}(k) \right)_{1 \leq i \leq m, 1 \leq j \leq n}$$

and the Hessian for each model component $M_i(k), 1 \leq i \leq m$, as follows:

$$D^2 M_i(k) = \left(\frac{\partial^2 M_i}{\partial k_l \partial k_j}(k) \right)_{1 \leq l \leq m, 1 \leq j \leq n}$$

Then the gradient and the Hessian of j read:

$$\nabla j(k) = DM^T(k)M(k)$$

$$H_j(k) \equiv D^2 j(k) = DM^T(k)DM(k) + \sum_{i=1}^{m} M_i(k)D^2 M_i(k)$$

Let us remark that in the *previous linear case*, the gradient and the Hessian read:

$$\nabla j(k) = A^T M(k) = A^T(Ak - z) \text{ and } H_j(k) = A^T A$$

since the term $D^2 M_i$ in the Hessian expression vanishs.

Gauss-Newton, Levenberg-Marquardt methods

The *Newton method* applied to the optimality condition $\nabla j(k) = 0$ reads at each iteration:

$$H_j(k^n) \cdot \delta k = -\nabla j(k^n) \; ; \quad k^{n+1} = k^n + \delta k$$

Newton's method requires the computation and the inversion of the Hessian of j.
For complex real-world models, this feature is often prohibitive because too complex or too CPU-time consuming.

The principle of the *Gauss-Newton method* is to consider in the Newton method the linearized Hessian (i.e. its approximation omitting the second order term):

$$H_j(k) \approx DM^T(k)DM(k)$$

It gives at each iteration:

$$\boxed{DM^T(k^n)DM(k^n) \cdot \delta x = -DM^T(k^n)M(k^n)} \tag{1.6}$$

Then the linear system to be inverted is symmetric positive, and definite if $DM(k^n)$ is full rank.
The Gauss-Newton method is observed to be very efficient if $DM^T(k^n)$ is full rank and if $M(k)$ small when close to the solution. On the contrary it becomes inefficient if these two conditions are not satisfied and/or if the model is locally "far to be linear" i.e. if the term $\sum_{i=1}^{m} M_i(k)D^2 M_i(k)$ is dominant.

Finally, a good alternative method to solve non-linear least square problems is the *Levenberg-Marquardt algorithm*, see e.g. [3].
This algorithm is somehow a damped version of the Gauss-Newton method. It can be perceived as a combination of a descent algorithm in a trust region and the Gauss-Newton algorithm too.

In summary, the most classical algorithms to solve "explicit" least square problems, linear or not, have been recalled. VDA algorithms are based on optimal control methods; the latter are based on the minimization of a criteria j too; somehow, in the least square sense... However, on the contrary to the "explicit" least square problems above, the unknown k is based on an underlying model; the physically-based model is an additional constraint to the optimization problem. As a consequence, the optimal solution k^* cannot be explicitly determine as presented in this introduction.

1.2.4 Rank deficience and SVD *

This is a "To go further section".

In the case A rank deficient ($\text{rank}(A) = r < m$), solutions exist but are not unique. Recall that if k is solution then $(k + y), \forall y \in Ker(A)$, is solution too. In this case, the functional j is convex, but not strictly convex since A^TA is not definite.

Nevertheless, it can be easily proved that it exists an unique solution k^* to the normal equations in $Ker^T(A)$ [2].
The right mathematical tool to analyze such ill-posed linear inverse problem is the Singular Value Decomposition (SVD).

Recalls on the SVD. Given a rectangular $m \times n$-matrix A, $\text{rank}(A) = r < m$, the SVD of A reads:

$$A = U\Sigma V^T = \sum_{i=1}^{r} \sigma_i u_i v_i^T$$

where:

- Σ is the $r \times r$-diagonal matrix containing the singular values σ_i of A:

$$(\sigma_i)^2 = \lambda_i(A^TA) = \lambda_i(AA^T) \text{ for all } \lambda_i \neq 0,\ 1 \leq i \leq r$$

$$0 < \sigma_r \leq ... \leq \sigma_1$$

- $U = (U_1, ..., U_r)$, $m \times r$-matrix, contains the unit orthogonal eigenvector basis of AA^T: $(AA^T)U = U\Sigma^2$ with $U^TU = \mathcal{I}_r$,
- $V = (V_1, ..., V_r)$, $n \times r$-matrix, contains the unit orthogonal eigenvector basis of A^TA: $(A^TA)V = V\Sigma^2$ with $V^TV = \mathcal{I}_r$.

The vectors of U constitute an unit orthogonal basis of $Im(A) \subset \mathbb{R}^m$, while the vectors of V constitute an unit orthogonal basis of $Ker^T(A) \subset \mathbb{R}^n$.

From the SVD, a *pseudo-inverse* (or generalized inverse) of the rectangular matrix A can be defined as follows:

$$A^{-1} = V\Sigma^{-1}U^T$$

The least-square solution in terms of SVD. Let us recall that since A is rank deficient, $\text{rank}(A) = r < m$, the calibration problem admits solutions but non-unique. Nevertheless, it exists an unique solution k^* presenting a minimal norm $\|x\|_2$ (and satisfying of course the calibration problem "$Ak = z$ at best"). In other words, the following problem has an unique solution k^*:

$$\min_{k \in \mathbb{R}^n \text{ s.t. } A^TAk = A^Tz} \|k\|_2$$

[2] If necessary, the reader should consult his favourite course on the topic, or any excellent book / course available online.

Furthermore, it can be proved that [3], [4] :

$$\boxed{k^* = A^{-1}z = V\Sigma^{-1}U^T z = \sum_{i=1}^{r}(\sigma_i)^{-1}u_i^T v_i z}$$

with the singular values σ_i ordered as mentioned previously i.e. decreasing gradually to 0.
The expression of k^* shows that the more the number of singular values are taken into account (from 1 to r at maximum), the more the data errors are amplified. In practice, while performing a SVD analysis of an ill-posed linear inverse problem, a key-point is to consider the "right number" of singular values; not too much, not too few...

1.3 From the control of dynamical systems to data assimilation

Let us present now the connection between the optimal control of differential equations and the so-called Variational Data Assimilation (VDA) approach.

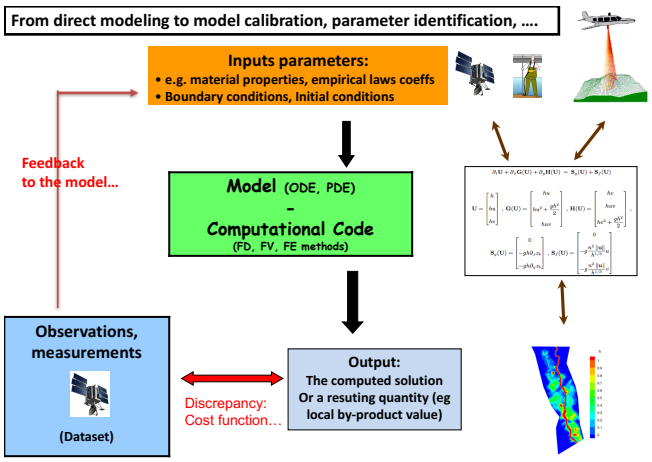


Figure 1.8: Step 1: Direct modeling and comparison of the numerical model outputs with measurements-observations-data. Step 2: towards a feedback to the model...

1.3.1 On optimal control

The control theory aims at analyzing dynamical systems modified by a command (the control).

The two objectives of the control theory are the followings:
A) Acting on the control in order to bring the system state to a final state given (if possible). It is a *controllability* problem.

[3] Keypoint for the proof: show that $(A^T A)A^{-1}z = A^T z$.
[4] The reader can find a very nice illustrated summary of the SVD and their application to inverse problems in the course online written by H.E. Krogstad, IMF, Norway.

The goal is generally to stabilize the system in order to make it insensitive to perturbation. This approach is not considered in the present course since not directly related to Data Assimilation.

B) *Defining a control u such that a criteria (the cost function) j is minimal.*
The cost function j depends on the control and the state of the system: it is an *optimal control* problem.

Generally the control must satisfy constraints.
Optimal control is a science between the automatic science and applied mathematics.

The Variational Data Assimilation (VDA) approach is based the optimal control of differential equations (the physically-based model).

In the next chapters, the optimal control of systems will be studied. First, the mathematical models will be Ordinary Differential Equation (ODE, dynamical systems), see Chapter 2. One focuses on the important *Linear-Quadratic (LQ) case*: the model is linear, the cost function is quadratic.
Next, the model is an elliptic Partial Derivatives Equations (steady-state PDE models), see Chapter 3. Finally, the considered model is an unsteady PDE models (parabolic or hyperbolic PDE systems), see Chapter 5.
VDA approach can be applied to EDS too; this mathematical equation type is not addressed in the present course.
Recall that all these classes of models represent an extremely wide range of systems encountered in engineering (and in academic researches) e.g. in fluid mechanics (geophysical or not), structural mechanics, micro-electronics, nano-technologies, biological systems, coupled multi-physic systems etc.

Calculus of variations deals with the minimization of functionals (i.e. mappings from a set of functions to $\mathbb{R}$; functionals are often definite integrals involving functions and their derivatives). Optimal control theory is somehow an extension of the calculus of variations: it is mathematical optimization problems with an underlying model (the differential equation). The goal is to derive a control policy.

To correctly attempt this course, students will need to be confortable with the basics of differential calculus. Mathematical recalls and exercises are proposed in Appendix.

Historically, optimal control appeared after the second world war with applications in aeronautics (missile guidance). A key point of the optimal control is the *Pontryagin minimum principle* (L. Pontryagin (blind) Russian mathematician (1908-1988)). The latter provides a necessary condition of optimality for an ODE system.

A simple example of optimal control (ODE model)

Let us consider a vehicle (robot, space vehicle) represented by the function $x(t)$. Typically, x represents its position and its velocity. The goal is to control the trajectory by acting on a command $u(t)$, its engine power and/or its wheel direction for example. We have: $x : [0, T] \to \mathbb{R}^n$ and $u : [0, T] \to \mathbb{R}^m$.

The mechanical laws provide the vehicle trajectory as follows:

$$\left\{ \begin{array}{l} \text{Given } u(t) \text{ and } s(t), \text{find } x(t) \text{ which satisfies:} \\ x'(t) = A(t)x(t) + B(t)u(t) + s(t) \text{ for } t \in I = [0,T] \\ \text{with the initial condition: } x(0) = x_0. \end{array} \right. \tag{1.7}$$

where A, B are two linear mappings defined from $I = [0,T]$ onto $M_{n,n}(\mathbb{R})$, $M_{n,m}(\mathbb{R})$, $s(t)$ is an external force (source term given in $\mathbb{R}^n$). For a sake of simplicity, the model considered is a 1st order linear ODE. Given $u(t)$, this problem has an unique solution $x(t)$.

Next, the *inverse problem* (optimal control problem here) is as follows.
Given a target trajectory $z(t)$, can we find (and compute) a control such that the vehicle is as close as possible to $z(t)$, moreover by consuming a minimum of power ?
To formulate this question in a mathematical point of view,the following cost function is defined:

$$j(u) = \frac{1}{2}\int_0^T \|x^u(t) - z(t)\|_W^2 dt + \frac{1}{2}\int_0^T \|u(t)\|_U^2 dt$$

with x the unique solution of the (direct) model, given u.
W and U are two symmetrical positive matrices, U definite, thus defining semi-norm and norm respectively.
The optimal control problem reads:

$$\min_{u(t)\in K} j(u)$$

where K is a convex closed subset of $\mathbb{R}^m$, including constraints on the control values (e.g. the power is bounded etc).

The cost function j depends on u explicitly through its second term, but also through $x(t)$ in its first term: this is why it is a *optimal control problem.*
The model operator $\mathcal{M}$ is defined as: $\mathcal{M}(u) = x^u$.

Note that the optimization formulation is bi-objective.

Since the model is linear, the cost function is strongly convex; this result is proved in next chapter. This problem is called a *Linear-Quadratic (LQ) optimal control problem.* The LQ problem is studied in Chapter 2.

1.3.2 From optimal control to Variational Data Assimilation (VDA)

Assimilating data (measurements, observations) into a model can be done using an optimal control process by *defining the cost function $j(k)$ as the misfit between the model output(s) and the measurements (the data).*

Generally in real-world DA problems, data are heterogeneous in space and time, and even in nature. Moreover, data are generaly very sparse and/or unfrequent. As a consequence, they may measure very partially only the modeled phenomena.
Recall that *numerical simulation*, based on a mathematical model, is a fundamental step for a large

range of industrial processes - conceptions - designs or physical - geophysical analysis - descriptions - predictions. Numerical simulation replaces more and more (real) experiments. Experiments are often built up for validation of the (virtual) numerical results only.

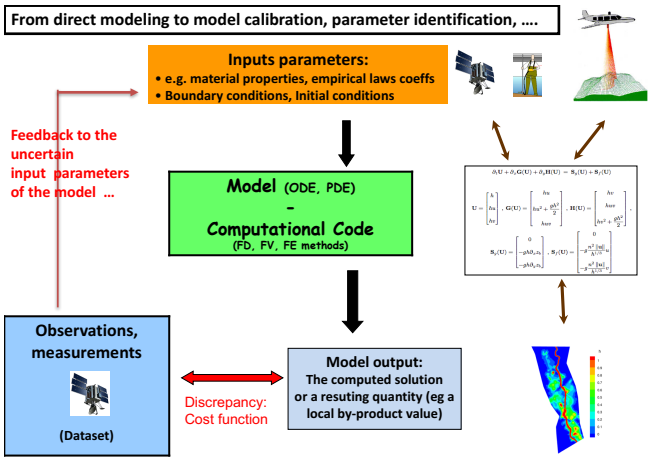


Figure 1.9: Goal of VDA: making fit the model output with data (observations, measurements) by "controlling" - identifying - calibrating some uncertain input parameters.

Two classes of Data Assimilation (DA) approaches

Data assimilation is the science combining "at best" knowledges of a system through: the model (mathematical equations), partial observations-measurements of reality and prior statistical errors - probabilistic behavior (eg through the employed metrics).

DA refers to different and complimentary approaches, see e.g. [14]:

1) A purely stochastic approach: *sequential filters* e.g. the Kalman filter computing the Best Linear Unbiased Estimate BLUE, the Ensemble Kalman Filter (EnKF).

2) The VDA approach based on the *control theory* It is a deterministic approach since optimizing a cost function while respecting a physical-based model constraint. However priori stochastic information can be introduced in the process eg through the employed metrics.A cost function measuring the misfit between the model output and the measurements is minimized.
Each approach presents advantages and drawbacks. The present course focuses on the VDA approach (based on the adjoint equations); the sequential filter approaches are not studied in details. However, it will shown that in the LQ case, both approaches are equivalent.

Calibrating models: what for ?

Calibrating a model can be done for different goals.
The goal can be to *identify* uncertain-unknown parameters values.

In the case of a time-dependent model, one may calibrate the model from past measurements (observations); next once calibrated, the model is a-priori more reliable, more accurate for *prediction*...

Typically this is what is done every 6H to predict the weather (for 5 days forecast), see a description of this rich and complex DA problem in Appendix.

Building up a model able to describe "at best" available observations is cheering before using it to predict, estimate unobserved values.
However, a robust calibration process does not ensure that the resulting model is reliable for prediction ! Typically the calibrated model should not be used out of range of its calibration value set. This remark fully applied to supervised learning in machine learning too... In practice, VDA method is very CPU time consuming: 10-100 times the CPU time of one direct simulation. However, it enables to greatly improve the model accuracy; they enable to improve the modeler understanding of the physical system too.

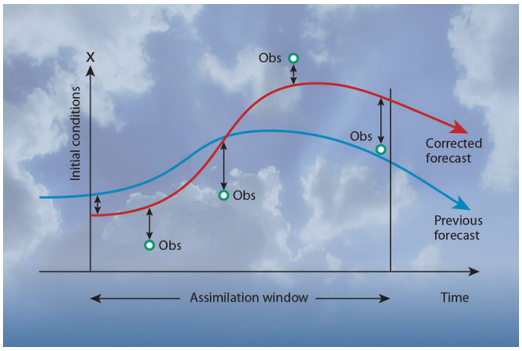


Figure 1.10: One of calibration goal: setting up a "good" model for prediction....(Image extracted from a ECMWF course).

Some history *Data assimilation* is a relatively recent science. For large scale problems (in the sense involving huge data amount), DA methods should be considered as a standard feature of numerical modelling.

Least-square method is a standard method to approximate a solution of over-determined systems. Historically, it has been elaborated by J.C. Gauss (1777-1855) and A.-M. Legendre (1752-1833). J.C. Gauss calculated (at 24 years old...) a correct prediction of an asteroid trajectory based on past observations.

Somehow, VDA solves *non-linear least-square problems* with an underlying physically-based model: the physically-based model constitutes a constraint of the optimization problem.

The *Kalman filter* (sequential approach) has been developed in the 60's by R. Kalman[5]. These se-

[5]R. Kálmán, Hungarian-born American electrical engineer and mathematician, inventor of the Kalman filters

quential filters have been used in particular by NASA during the Apollo program to better estimate the trajectories.

The historical application of DA and more particularly VDA, is the weather forecast problem (atmosphere dynamics). This extremely complex problem is briefly presented in appendix.

An example of VDA (PDE model)

Please read the topic of the proposed programming practical in next section.

DA course part outline This course follows the outline below.
Chapter 3 aims at deriving the optimality systems for elliptic PDEs (a-priori non-linear) and focus on the computational aspects. A result of existence and uniqueness of the control is presented in the LQ PDE case. We present in detail how to solve numerically the optimality system. The final goal is the *identification* of the uncertain / unknown input parameters; in other words a better *calibration* of the model.

In Chapter 5, the method is extended to unsteady PDEs systems (either parabolic or hyperbolic). Calculations derived in these cases are more formal. The classical computational algorithms is presented; it is sometimes the "4D-var" algorithm.

In Chapter 4, the equivalence between the present VDA approach and the Best Linear Unbiased estimator (BLUE), also the sequential Kalman filter, also the most probable solution (Bayesian view) are demonstrated in simple examples. These equivalence are valid for Linear-Quadratic (LQ) problem that is the model is linear, the observation operator is linear therefore the cost function is strictly convex and it admits an unique minimum.

A few references The reader can find many references related to the present multi-topic course. Concerning VDA, let us cite a few pioneer works: Y. Sasaki in the ∼60-70's [37, 38], F.X. Le Dimet - O. Talagrand [27]; see also [21, 15, 31] plus the numerous historical cited references therein.
The course attendants may consult the books [7, 8], the following historical online course [10] (and [19] if more interested in sequential filters). Up-to-date surveys are available in numerous excellent PhD thesis manuscripts too.

Acknowledgments

The author is very grateful to all his current and former collaborators: PhD students, research engineers and post-doctoral researchers. We've worked together on VDA, see the references indicated on the computational software webpage [33]. All these colleagues have greatly contributed to the author's understanding and to the material presented in this manuscript.

1.4 Connections with Artificial Neural Networks (Machine Learning)

It is interesting to notice that "supervised learning" applied to physically-based data can be an alternative approach to empirically build up an effective model and/or to solve identification problems.
In this sense, ML may be viewed as "data assimilation" not relying on a physical model (eg equations based on law conservations), but relying on an empirical effective model: an Artificial Neural Network (ANN).

The principle feature of an ANN-based model is its architecture. The latter defines the multi-scale resolution, non-linear resolution features. ANNs (and their numerous versions like CNN, RNN etc) enable to find nonlinear trends between data.

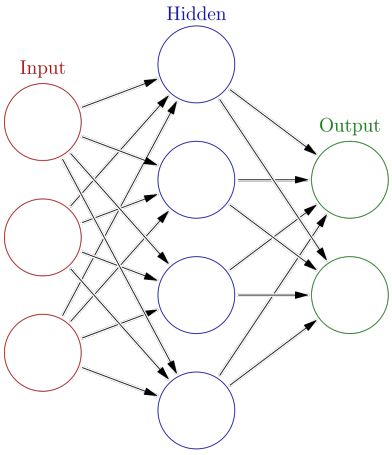


Figure 1.11: An Artificial Neural Network (ANN) with a single hidden layer. (Image extracted from Wikipedia page).

Let us summarize very briefly how an ANN can be built up, trained from (large) datasets, and finally how it may be employed for prediction.

A large training dataset $\mathcal{D}$ describing the targeted phenomena has to be given. $\mathcal{D}$ contain "examples" (samples) $(I_i, O_i)^{obs}$, $i = 1, \cdots, m$ with I_i the i-th input variable and O_i the corresponding output.

In the present course context and if adopting the same notations, the operator $I \mapsto O$ may be either the underlying direct model or the underlying inverse model, depending on the point of view. That is, one may consider (depending on the point of view):

$$\mathcal{M}(\mathbf{k};\cdot) : I \mapsto O_{(\mathbf{k};I)} \text{ or } \mathcal{M}^{-1}(\mathbf{k};\cdot) : I \mapsto O_{(\mathbf{k};I)}$$

The learning stage aims at producing the desired output for each input.
In other words, it consists to built up the *unknown underlying model* $\mathcal{M}$ (or $\mathcal{M}^{-1}$, see above).

In ML terminology, the mapping operator above is the called the *estimator*.

The network consists of connections between perceptrons. Each connection is assigned to a weight value $k_{i,j}$, measuring the connection importance.
The estimator $\mathcal{M}$ (or $\mathcal{M}^{-1}$, see above) can be determined as the minimizer of the following typical differentiable cost function (also called loss function):

$$j_{\mathcal{D}}(\mathbf{k}) = \frac{1}{m}\sum_{i=1}^{m}\left(O_i^{obs} - \mathcal{M}(\mathbf{k}; I_i^{obs})\right)^2 = \|O^{obs} - \mathcal{M}(\mathbf{k}; I^{obs})\|_{2,m,h}^2 \tag{1.8}$$

The optimization variable $\mathbf{k}$ is constituted by the ANN weight parameters. Each weight k_j is a matrix of dimension $n_{out} \times n_{in}$, $n_\square$ the connections number.
The resulting optimization problem is of huge size:

$$\min_{\mathbf{k}} j_{\mathcal{D}}(\mathbf{k})$$

The training step consists to solve this optimization problem i.e. to identify the optimal values of the ANN parameters $k_j,\ j = 1, .., Nlay$. $Nlay$ is eg. equal to the number of layers. ANN are efficient if built up from numerous hidden layers: this is *deep learning*.
In the end, the "ANN model" is determined by its architecture and its weight parameters $(k_1, \cdots, k_{Nlay})$, see Fig. 1.11.

For each perceptron, the most efficient activation function is the rectified linear unit (ReLU) function. The "hyper-parameters" of the algorithm (learning rate, decay rate, dropout probability) can be viewed as "priors" of the model. They are experimentally chosen... The selected values are those providing the minimal value of $j(\cdot)$.

To minimize $j_{\mathcal{D}}(\mathbf{k})$ a first-order gradient-based stochastic optimization is often employed eg the classical Adam method.
The cost gradient is computed by the so-called "back-propagation" procedure. It is simply the application of the derivative of (numerous) function compositions. This important technical step can be performed using "Automatic Differentiation" like those presented in Appendix.

After the training phase, one obtain an empirical effective model (not really understood yet...).
The latter may be employed as a predictor, however the ANN model thus built is very likely valid within the learning range values only...
Nowadays, an ANN can been coded in Python very easily eg. using PyTorch - Mpi4Py libraries, TensorFlow etc.

1.5 Programming practical in Earth sciences

Below is presented a moderately complex example. This example will be developed along the course. A Python code solving the problem and assessing the employed VDA method is provided on the INSA Moodle page. This programming practical has been designed in collaboration with Thibault Malou, PhD student at INSA & CLS corp. Moreover, Thibault Malou has programmed the solution code. For the INSA students, T. Malou is the supervisor of this programming practical.

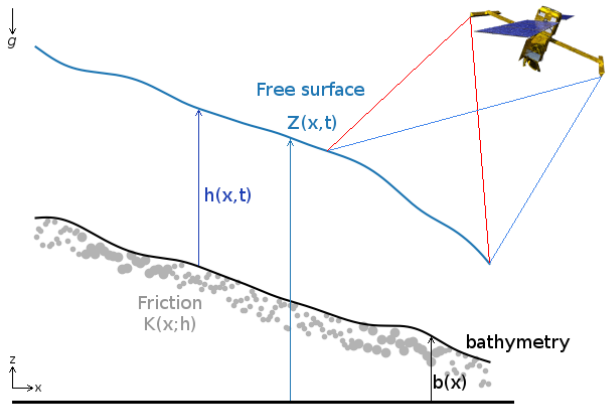


Figure 1.12: The inverse problem arising in spatial hydrology: identifying river features (unobserved bathymetry $b(x)$, friction parametrization $K(x; h(x,t))$ and discharges values $Q(x,t)$) from large scale and sparse altimetry measurements: water surface elevation $H(x,t)$ and river width $w(x,t)$

1.5.1 The direct model: river flow dynamics

River flow is a subject widely studied especially from a (every) long time in particular with the Saint-Venant equations (also known as the 1D shallow water equations). *(Barré de Saint-Venant, 1797-1886, French "ingénieur", physicist and mathematician).*
The (only) dimensional number of the Saint-Venant equations is the Froude number denoted by Fr. At large scale, river flows presents in great majority low Froude numbers : $Fr <\approx 0.3$ and lower. This leads to subcritical flows, no shock can occur.
The flow is considered 1D; the river geometry is defined as indicated on Fig. 1.13. The wetted cross-sections $A(x,t)$ (m^2) are represented by rectangles of width $w(x,t)$ (m).
We denote: $H(x,t)$ the free surface height, $b(x)$ the bathymetry, $h(x,t)$ the water depth and $u(x,t)$ the (depth-averaged) velocity.
The subcritical flow hypothesis ($Fr < 1$,$Fr = \frac{u}{\sqrt{gh}}$) implies that the mass and momentum equations can be combined into a single equation; this leads to the so-called *diffusive wave model (lubrication theory).*
The flow model reads as follows:

$$\partial_t H(x,t) - \Lambda(u,H,b;x,t)\partial^2_{xx} H(x,t) + c(u;x,t)\partial_x H(x,t) + c(u;x,t)\frac{\partial_x w(x)}{w(x)} H(x,t) \quad (1.9)$$

$$= \; c(u;x,t)\partial_x b(x) + c(u;x,t)\frac{\partial_x w(x)}{w(x)} b(x) \quad (1.10)$$

with :

$$\Lambda(u,H,b;x,t) = \frac{1}{2} u(x,t)\frac{(H(x,t)-b(x))}{|\partial_x H(x,t)|} \quad (1.11)$$

the effective wave diffusion and

$$c(u;x,t) = \frac{5}{3} u(x,t) \quad (1.12)$$

the wave velocity.

Spatial hydrology combines the well known hydrology science with large scale measurements from space.
The crucial flow feature is the discharge $Q(x,t)$: $Q(x,t) = u(x,t)A(x,t)$ (m^3/s).

Altimetry measures free surface height $H^{in}(t)$, resp. $H^{out}(t)$ at very large ($\approx$ 1 km long) and with some uncertainty ($\pm$10 cm at 1 km scale).

The direct model consists to determine the water flow height $H(x,t)$ solution of (1.10) (accompanied with initial condition and Dirichlet boundary conditions), given the bathymetry $b(x)$, the width $w(x)$ and the velocity $u(x,t)$.

1.5.2 The spatial hydrology inverse problem

Spatial altimetry provides free surface height data (denoted by H^{obs}) at large scale (1km long), on several points along the river with a few days frequency.

In practice, several issues remain in solving the direct model (1.10):
- the river bathymetry is in great majority unknown and there is no way to measure it everywhere on Earth yet.
- the (depth-averaged) velocity is unknown (may be excepted at very very locations). Again, there is no way to measure it everywhere, at all time yet.

Then the inverse problem to be solved reads:

Given some sparse, large scale and unfrequent measurements of the free surface height $H^{obs}(x,t)$ provided by satellite altimetry, estimate the flow velocity $u(x,t)$ and the bathymetry $b(x)$.

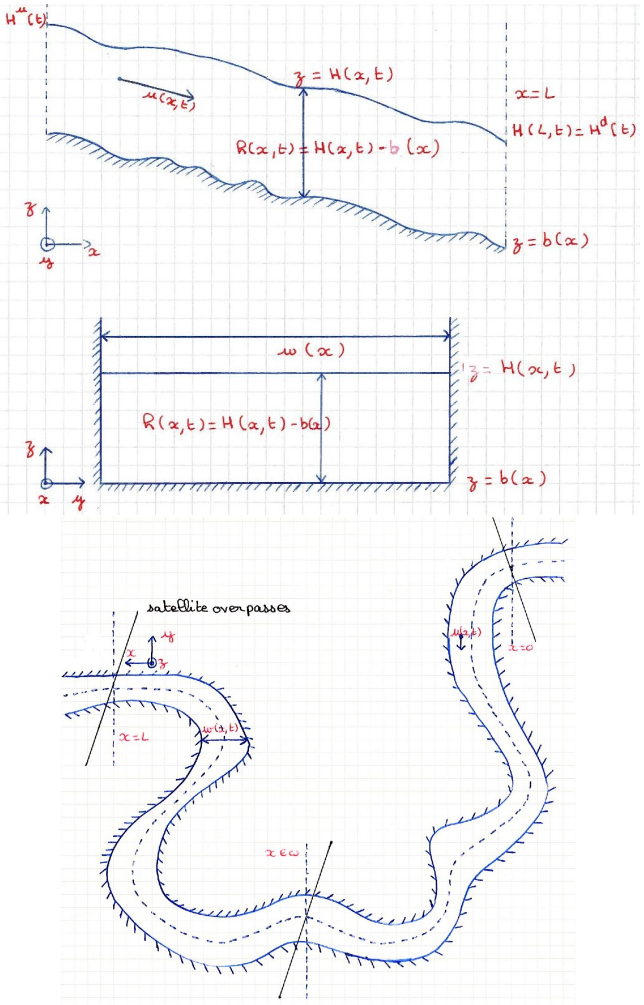


Figure 1.13: River geometry and notations

In the VDA formalism, the "unknown input parameter" is:

$$k(x,t) = (b(x), u(x,t)) \tag{1.13}$$

while the model output is $H(x,t)$.

Note that the altimetry instrument measure directly the model output quantity H; this not the case in many real-life problems.
Indeed, it is usual to have to introduce a more or less comple observation operator mapping the model output to the observation (e.g. a velocity to a signal).

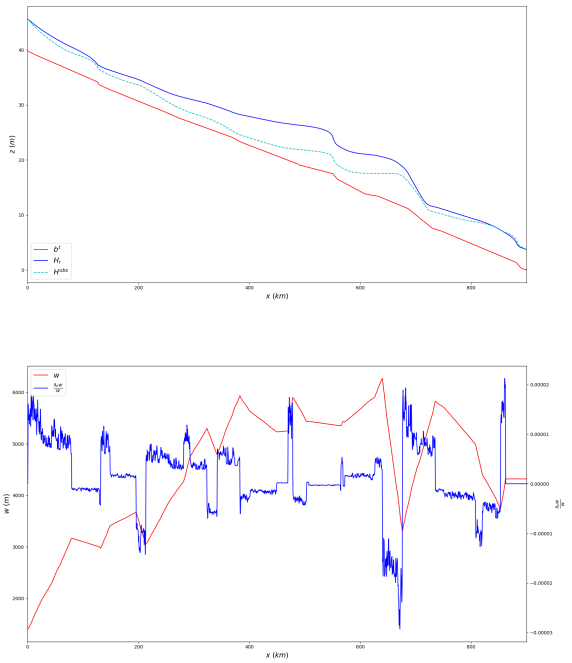


Figure 1.14: A forward run in the following case: stationnary, linearized flow model in the vicinity of the Water Surface (WS) H_r. (L) Vertical view: the WS $H(x)$, $H_r(x)$ and the bathymetry to identify $b(x)$. Top view: the river width $w(x)$ and the model term value $(w'/w)(x)$. Images: output of your practical computational code developed by T. Malou, INSA.

Chapter 2

Optimal Control of ODE: the Linear-Quadratic (LQ) case

In this chapter, the optimal control of systems governed by an Ordinary Differential Equation is briefly studied; we refer for example to [18, 41] for a full course on the subject. The present chapter follows closely the presentation done in [41].

The basic and important case is studied: the case with a linear ODE model and a quadratic cost function; this the so-called "LQ case". A proof of existence and uniqueness of the optimal solution is given. Next its characterization using the Pontryagin's minimum principle and the Hamiltonian is presented.
Also the notion of controllability and the Kalman's condition are briefly presented (these concepts may be skipped).

The outline of this chapter is as follows[1]

Contents

[1]Recall that the sections indicated with a * are "to go further sections". They can be skipped in a first reading or if the reader is not particularly interested in deeper mathematical basis, mathematical proofs.

2.1 Introduction

In order to illustrate what is an optimal control problem let us give a simple example (extracted from a Wikipedia webpage).
Consider a car traveling on a straight line through a hilly road. The question is, how should the driver press the accelerator pedal in order to minimize the total traveling time? The control law refers specifically to the way in which the driver presses the accelerator and shifts the gears. The "system" or "direct model" is the car dynamics, and the optimality criterion is the minimization of the total traveling time. Control problems usually include additional constraints. For example the amount of available fuel might be limited, the speed limits must be respected, etc.
The "cost function" is a mathematical functional giving the traveling time as a function of the speed, initial conditions and parameters of the system.

Another optimal control problem can be to find the way to drive the car so as to minimize its fuel consumption, given that it must complete a given course in a time not exceeding some amount. Yet another control problem can be to minimize the total monetary cost of completing the trip, given assumed monetary prices for time and fuel.
Let us notice that the optimal control problem may have multiple solutions (i.e. the solution may not be unique), if it exists...

2.2 Illustrative example

The aim of this section is simply to illustrate (or recall) how the response of a apparently gentle ODE equation may widely differ depending on the source term or the applied command.

2.2.1 The controlled system

This classical example is treated in detail in [41], we refer to this book for more details

Let us consider a spring - mass system (see Figure 2.1). The mass m is submitted to a force $f(x)$, which supposed to be equal to:$-[k_1(x-L)+k_2(x-L)^3]$. L is the spring length at rest, k_* are the spring stiffness parameters.

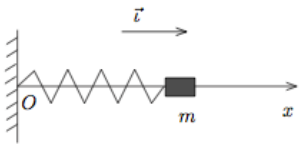


Figure 2.1: The spring example.

We will apply an external force: $u(t)\vec{\imath}$ (depending on time t).
Given the external force, the spring position $x(t)$ is described by the equation:

$$mx''(t) + k_1(x(t) - L) + k_2(x(t) - L)^3 = u(t) \text{ for } t \geq 0$$

It is the *direct model*.It is an ordinary differential equation, linear if $k_2 = 0$, non-linear if not.
The initial condition, $x(0) = x_0, x'(0) = y_0$, is given.

The problem we consider is as follows. Given an initial state (x_0, y_0), find the "best" external force $u(t)$ such that $x(t) = L$ in a minimal time, under the constraint $u(t) \leq 1$ (the external force is bounded).

$u(t)$ is the *control* of the problem.

For sake of simplicity (and without any change of the nature of problem), we set: $m = 1, k_1 = 1, L = 0$.
Then in the phase space (x, x'), the direct model reads:

$$X'(t) = AX(t) + f(X(t)) + Bu(t) \ , \quad X(0) = X0.$$

where $X(t) = (x, x')(t)$ is the *state* of the system,

$$A = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \ , \quad B = (0, 1)^T \ , \quad f = (0, -k_2 x^3)^T$$

The model is a 1st order differential system, linear if $k_2 = 0$, non-linear if not.

2.2.2 Different states of the system according to the control values

1) In the case $u(t) = 0$ (no external action), the solution of the system satisfies:

$$x''(t) + x(t) + 2x(t)^3 = 0$$

It is a particular case of the Duffing equation. Its solution $x(t)$ satisfies: $x(t)^2 + x(t)^4 + x'(t)^2 = cste$.
All solutions are periodic and can be represented by an algebraic curve. The phase diagram and the trajectory are plotted on Fig. 2.2.

2) In the case $u(t) = -x'(t)$, i.e. we try to damp the spring, the model reads:

$$x''(t) + x(t) + 2x(t)^3 + x'(t) = 0$$

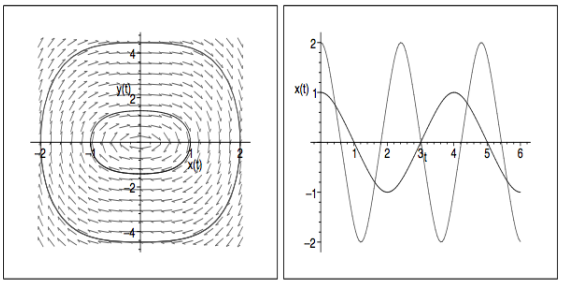


Figure 2.2: The spring example. State of the system without control, from [41].

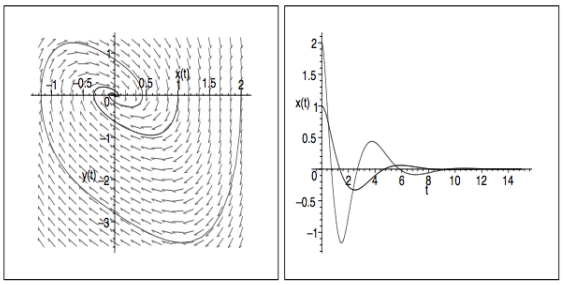


Figure 2.3: The spring example. State of the system with damp control, from [41].

Its numerical solution is computed, the corresponding phase diagram and the trajectory are plotted on Fig. 2.3.

Using the Lyapunov theory, it can be shown that the origin is asymptotically stable. The spring position and the velocity reach the equilibrium position in infinite time, not in finite time.
Therefore this control do not satisfies the original problem.

3) Let us set $u(t) = -(x(t)^2 - 1)x'(t)$. With this particular control expression, the model reads:

$$x''(t) + x(t) + 2x(t)^3 + (x(t)^2 - 1)x'(t) = 0$$

It is a particular case of the *Van der Pol* equation.

Two different solutions are computed and plotted on Fig. 2.4 (phase diagram and trajectories).

Using the Lyapunov theory, it can be proved that there exists a periodic solution which is attractive (Figure 2.4), [41].

These three examples simply illustrate the wide range of behaviors which can be obtained from a very simple differential system by changing the control term.

2.3 Controllability *

This is a "To go further" section...

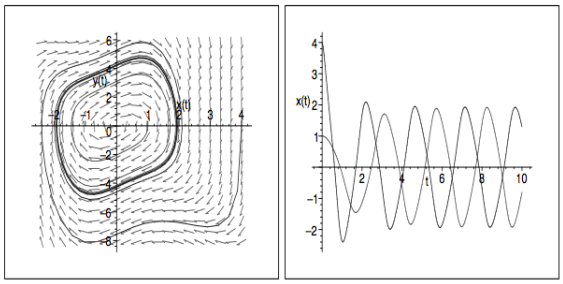


Figure 2.4: The spring example. State of the system with a given control (Van der Pol like equation), from [41].

This section presents the basic concept of controllability of a dynamical system and the Kalman's condition. These concepts are important in a context of an automatic course, and more secondary in the present Data Assimilation concept. Therefore this section may be skipped if the reader wants to focus on VDA only.

2.3.1 Reachable sets

Let $x_u(t)$ be the solution of the ODE system corresponding to a given control $u(t)$ (x_u is supposed to exist and to be unique).

Definition 2.1. *The set of reachable states from the initial state x_0 in time $T > 0$, is defined by:*

$$Acc(x_0, T) = \{x_u(T), \text{ with } u \in L^\infty([0,T], \Omega)\}$$

with Ω a compact subset of $\mathbb{R}^m$.
We set: $Acc(x_0, 0) = x_0$.

Theorem 2.2. *Let us consider the following 1st order linear ODE system in $\mathbb{R}^n$:*

$$x'(t) = A(t)x(t) + B(t)u(t) + r(t)$$

Then $Acc(x_0, t)$ is compact, convex, and varies continuously with t, $t \in [0, T]$.

Partial proof. We prove only that $Acc(x_0, t)$ is convex in the case $u \in L^\infty([0,T], \Omega)$ with Ω *convex.* We refer to [41] for the proof of the remaining results.

Let Ω be convex. Let $x_1^t, x_2^t \in Acc(x_0, t)$; we denote by u_i the corresponding controls. We have:

$$x_i(0) = x_0 \ ; \ ; \ x_i'(s) = A(s)x_i(s) + B(s)u_i(s) + r(s) \ ; \quad i = 1, 2$$

We denote: $x_i^t \equiv x_i(t)$. Let $\lambda \in [0, 1]$. We seek to prove that: $(\lambda x_1^t + (1-\lambda)x_2^t) \in Acc(x_0, t)$.
We solve the 1st order linear ODE, we obtain:

$$x_i^t \equiv x_i(t) = M(t)x_0 + M(t)\int_0^t M(s)^{-1}B(s)u_i(s)ds$$

with $M(t) \in M_{n,n}(\mathbb{R})$ such that: $M'(t) = A(t)M(t)$, $M(0) = Id$.
Note that if $A(t) = A$ constant then $M(t) = \exp tA$.

We set: $u(t) = \lambda u_1(t) + (1-\lambda)u_2(t)$. Since Ω is convex, $u(t) \in \Omega$; also $u \in L^\infty([0,T],\Omega)$.

Let $x(t)$ be the solution associated to $u(t)$. We have $x(t) \in Acc(x_0,t)$ and:

$$x(t) = M(t)x_0 + M(t)\int_0^t M(s)^{-1}B(s)u(s)ds$$

In other respect,

$$\begin{aligned}\lambda x_1^t + (1-\lambda)x_2^t &= M(t)x_0 + \lambda[M(t)\int_0^t M(s)^{-1}B(s)u_1(s)ds] + (1-\lambda)[M(t)\int_0^t M(s)^{-1}B(s)u_2(s)ds] \\ &= x(t)\end{aligned}$$

Thus $(\lambda x_1^t + (1-\lambda)x_2^t) \in Acc(x_0,t)$ and $Acc(x_0,t)$ is convex. □

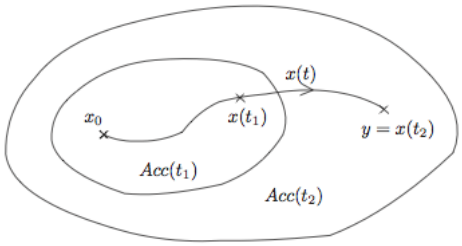


Figure 2.5: Reachable set in the linear case, from [41].

A more simple case

Let us consider: $r = 0$ and $x_0 = 0$. The system becomes:

$$x'(t) = A(t)x(t) + B(t)u(t)\ ;\quad x(0) = x_0$$

and its (unique) solution reads:

$$x(t) = M(t)\int_0^t M(s)^{-1}B(s)u(s)ds$$

Thus x is linear in u. Then we have

Proposition 2.3. *Let us consider the system:* $x'(t) = A(t)x(t) + B(t)u(t);\ x(0) = x_0$, *and* $\Omega = \mathbb{R}^m$. *Then for all* $t > 0$, *the reachable set* $Acc(0,t)$ *is a vectorial subspace of* $\mathbb{R}^n$.
If we assume in addition $B(t) = B$ *constant, then for any* $0 < t_1 < t_2$, *we have:* $Acc(0,t_1) \subset Acc(0,t_2)$.

We refer to [41] for the proof.

2.3.2 Controllability of autonomous linear systems: Kalman's condition

In the following we consider a first order linear autonomous ODE system in $\mathbb{R}^n$:

$$x'(t) = Ax(t) + Bu(t) + r(t)$$

with A and B independent of t. Furthermore, we set $\Omega = \mathbb{R}^m$ (i.e. we do not constraint the control variable).

We say that the system is controllable at any time T if $Acc(x_0, T) = \mathbb{R}^n$; it means that for any x_0 and x_1 in $\mathbb{R}^n$, there exists a control u such that $x(0) = x_0$ and $x(T) = x_1$.

Theorem 2.4. *Under the assumptions above, the first order linear autonomous ODE system is controllable at any time T if and only if the rank of matrix*
$C = (B, AB, ..., A^{n-1}B)$ *is equal to n.*

The matrix C is called the Kalman matrix and the condition $\text{rank}(C) = n$ is called *the Kalman controllability condition.*

Since the Kalman's condition does not depend neither on T nor on x_0, then the first order linear autonomous system is controllable at any time from any initial condition. (R. Kalman is a hongarian-american mathematician-electrical engineer, born in 1930).
Sketch of the proof. First, we proves that $\text{rank}(C) = n$ if and only if the following linear application Φ is surjective.

$$\Phi : L^\infty([0,T], \mathbb{R}^m) \mapsto \mathbb{R}^n \ ; \quad u \mapsto \int_0^T \exp\left((T-t)A\right) B u(t) dt$$

Thus $\Phi : L^\infty([0,T], \mathbb{R}^m) = \mathbb{R}^n$. In other respect, given u, we have:

$$x(T) = \exp(TA)x_0 + \int_0^T \exp\left((T-t)A\right)(Bu(t) + r(t))dt$$

Thus the reachable states is:

$$Acc(x_0, T) = \exp(TA)x_0 + \int_0^T \exp\left((T-t)A\right) r(t) dt + \Phi(L^\infty) = \mathbb{R}^n$$

Therefore, the system is controllable.
We refer to [41] for the full proof of the theorem

These results give an overview of notions of controllability for *linear systems*. We refer e.g. to [41] for more details and the study of non-linear systems.

2.4 The Linear-Quadratic (LQ) problem

This section is important in all the following since it introduces the reference inverse problem: the Linear-Quadratic (LQ) problem.

2.4.1 The Linear model

Let A, B and s be three mappings defined from $I = [0, T]$ into $M_{n,n}(\mathbb{R})$, $M_{n,m}(\mathbb{R})$ and $\mathbb{R}^n$ respectively. The three mappings are assumed to be bounded i.e. $L^\infty(I)$ (this assumption could be relaxed since

locally integrable would be sufficient). We consider the following linear 1st order ODE.

$$\left\{\begin{array}{l} \text{Given } u(t), \text{ find } x(t) \text{ such that:} \\ x'(t) = A(t)x(t) + B(t)u(t) + s(t) \text{ for } t \in I = [0,T] \\ \text{with the initial condition: } x(0) = x_0. \end{array}\right. \tag{2.1}$$

The function $u(t)$ is assumed to be in $L^\infty(I)$.
In other words, we consider a phenomena which can be modelled by this linear ODE. (2.1) is the *direct model*, $x(t)$ is the *the state* of the system, and $u(t)$ will be the *control* of the system.

Existence and uniqueness of the solution. We known (classical theorem of existence) that (2.1) has one and only one solution $x(t)$, $x(t)$ continuous from I into $\mathbb{R}^n$.
Let us remark that we known an explicit expression of x in an integral form; the unique solution $x(t)$ depends on $u(t)$ of course.

The general *optimal control problem* reads as follows: find a control $u(t)$ *minimizing* a given criteria (cost function) $j(u)$.

Often, additional constraint on the state are imposed; e.g. $x(t)$ must go from x_0 (the initial state) to x_1 in finite time.

A controllability problem would read as follows: given $x_1 \in \mathbb{R}^n$, find a control $u(t)$ such that $x(t)$ goes from x_0 (at initial time) to x_1 in finite time.
Courses of "Automatic" or "Optimal command" may focus on the controllability problems for linear systems. In the present course, we need to consider optimal control problems only, not controllability problems.

Remark 2.5. *- Since the previous spring academic example is modeled by a linear ODE in the case $k_2 = 0$, the results which follow will apply to such a case.*
- Among the historical optimal control problems, let us cite the so-called brachistochrone problem (end of the 17th century, from the greek words "brakhisto" (shorter) and "chronos" (time)) where the system is a simple 1d gravitational dynamic without friction, and j is time, from x_0 to x_1.

2.4.2 The Quadratic cost function

The choice of the cost function to be minimized is part of the problem definition. Since it will be minimized, convexity properties are expected.
Moreover if using computational gradient-based methods, differentiability properties will be required too. As a consequence, quadratic cost functionals are almost always chosen. Somehow, this set the inverse control problem as a least-square problem.

Recall that the quadratic property implies differentiability and strict convexity.

A typical generic "quadratic" cost function reads as follows:

$$\boxed{j(u) = \frac{1}{2}\int_0^T \|x(t)\|_W^2 dt + \frac{1}{2}\int_0^T \|u(t)\|_U^2 dt + \frac{1}{2}\|x(T)\|_Q^2} \tag{2.2}$$

The three terms are the time averaged cost of the state, the control and the final state (in the metrics W, U and Q respectively).

This functional j is a multi-objective cost functional.

The operators (matrices) Q, W and U are given in $M_{n,n}(\mathbb{R})$, $M_{n,n}(\mathbb{R})$ and $M_{m,m}(\mathbb{R})$ respectively; they define some (semi-)norms. They are symmetric positive; moreover U is definite.
Note that Q, W must be symmetric positive, but definite is not necessary. Indeed, for example, one can have the cost function minimal for a vanishing control.

Recall that: $\|\cdot\|_\square^2 =< \square\cdot, \cdot >$.

Note that j is quadratic in its primal variables $(x; u)$ and not really in u.

Since the observation functional is quadratic, and the model is linear, the natural functional space for control variable is $M = L^2([0,T], \mathbb{R}^m)$.
Let us point out that the a-priori natural space $C^0([0,T], \mathbb{R}^m)$ is not a Hilbert space...

Let us recall that for a matrix M symmetric positive definite, in vertu of Courant-Fischer's theorem (Rayleigh's quotient), there exists a constant $c > 0$ such that:

$$\forall v \in \mathbb{R}^m, \quad \|v\|_M^2 \geq c\|v\|^2$$

In other words, the (linear) operator M is coercive, uniformly in time.

Let us define the *"Model Operator"* $\mathcal{M}(u)$ as:

$$\begin{array}{rcl} \mathcal{M}: \ u(t) & \mapsto & x_u(t) \\ M = L^2(I, \mathbb{R}^m) & \to & C^0(I, \mathbb{R}^n) \end{array}$$

We have the important property:

Proposition 2.6. *The model operator $\mathcal{M}(u(t))$ is affine for all t in $[0,T]$.*

The proof is similar to Theorem 2.2 one; however let us recall its principles.
Let $x(t)$ be the (unique) solution associated to $u(t)$: $x(t) = \mathcal{M}(u(t))$.

In the present case, the model is a linear EDO; therefore one can write an explicit expression of its solution.
Indeed, we have:

$$x(t) = M(t)x_0 + M(t)\int_0^t M(s)^{-1}B(s)u(s)ds$$

with $M(t) \in M_{n,n}(\mathbb{R})$ such that: $M'(t) = A(t)M(t),\ \ M(0) = Id.$
Note that if $A(t) = A$ constant then $M(t) = \exp tA$.
The result follows straightforwardly from the explicit expression fo $x(t)$. □

The considered optimal control problem is a **Linear-Quadratic (LQ) optimal control problem**. It reads as follows:
Given x_0 and T, find $u(t)$ minimizing the "quadratic" cost function $j(u)$ defined by (2.2), with $x(t)$ solution of the linear EDO (2.1).

Remark 2.7. *The linear-quadratic problem seems a-priori idealistic; it is not so much. Indeed, observation functions are often defined quadratic in view to minimize differentiable functionals. Moreover, to better understand more complex non-linear problems, a good understanding of simplified linear model is often a necessary step.*

In the LQ case, many instructive properties of the system can be written, both in a mathematical and numerical point of view.

For a sake of simplicity, in the direct model we consider the source term $s(t) = 0$.

2.5 The Pontryagin principle & the Hamiltonian

In this section, the fundamental concepts and results are introduced.

2.5.1 Existence and uniqueness of the solution

First, let us prove the existence and uniqueness of the optimal control solution in the LQ case.

Theorem 2.8. *There exists an unique solution $u \in M$ minimizing $j(u)$, $j(u)$ defined by (2.2), with the "constraint" $x(t)$ solution of (2.1).*
In other words, it exists a unique optimal control $u(t)$ and a corresponding trajectory $x(t)$ to the LQ problem.

Proof. A) Existence ("To go further").
It is based on the convergence of minimizing sequence (calcul of variations, D. Hilbert, 1900 approx.).
Step 1). The cost function is bounded by below: $\inf\{j(u), u \in M\} > -\infty$ since $j(u) \geq 0$. There exists a minimizing sequence (u_n) defined for all $t \in [0, T]$; i.e. a sequence such that:

$$\lim_{n\to+\infty} j(u_n) = \inf\{j(u), u \in M\}$$

(As a matter of fact, $\forall n \in \mathbb{N},\ \exists v_n$ such that: $m \leq j(v_n) < m + \frac{1}{n}$).
Step 2) There exists $\alpha > 0$ such that: $j(u) \geq \alpha \|u\|_M^2$. Thus, the minimizing sequence (u_n) is bounded in M. Hence there exists a sub-sequence (u_{n_k}) which converges weakly to a control u in M:

$$u_{n_k} \rightharpoonup u \text{ in } L^2(I)$$

Step 3) Let us denote by x_n (resp. x) the state associated to u_n (resp. u). The system (2.1) is a first order linear O.D. E. (and with $s(t) = 0$); we known an explicit expression of the solution:

$$\forall t,\ x_n(t) = M(t)x_0 + M(t) \int_0^t M(s)^{-1} B(s) u_n(s) ds \tag{2.3}$$

with $M(t) \in M_{n,n}(\mathbb{R})$ such that: $M'(t) = A(t)M(t),\ \ M(0) = Id.$ (If $A(t) = A$ constant then $M(t) = \exp(tA)$).
Similarly, we have: $\forall t,\ x(t) = M(t)x_0 + M(t) \int_0^t M(s)^{-1} B(s) u(s) ds$. Thus, we obtain that the sequence $(x_n)_n$ converge to x.
Passing to the limit in (2.3), we obtain the existence of x_u, solution corresponding to u.
Step 4) It remains to prove that u minimizes j. Since $u_n \rightharpoonup u$ in L^2, since j is continuous hence lower semi-continuous, we have (by definition):

$$j(u) \leq \liminf_n j(u_n)$$

and necessarily $j(u) = \inf_{v \in M} j(v)$.

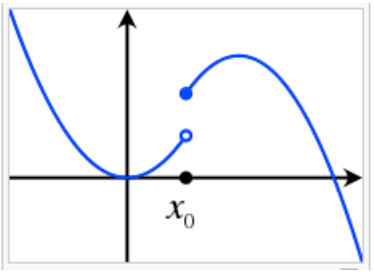


Figure 2.6: A function j lower semi-continuous at u_0: for u close to u_0, $j(u)$ is either close to u_0 or lower than u_0.

In other words, $u(t)$ minimizes $j(u)$ ($j(u) = \min_{v \in M} j(v)$) and the corresponding state (trajectory) x_u is an optimal trajectory.

B) Uniqueness. Let us prove that $j(u)$ is strictly convex i.e. $\forall (u_1, u_2) \in M^2,\ \forall t \in]0,1[$,

$$j(tu_1 + (1-t)u_2) < tj(u_1) + (1-t)j(u_2)$$

unless $u_1 = u_2$.

For all t, $\|u(t)\|_U$ is a norm hence convex but not strictly convex... Proof of this assertion: the triangle inequality.

However, in a Hilbert space, the square of a norm (eg $\|u(t)\|_U^2$) is strictly convex, see e.g. [6] chap. 10, p118.

In vertu of Lemma 2.6, the model operator $\mathcal{M}$ is affine therefore convex.
In other respects $\|.\|_W$ and $\|.\|_Q$ are semi-norms hence convex.

Finally the cost function $j(u)$ is strictly convex and the uniqueness follows straightforwardly. □

Note that the strict convexity of $j(\cdot)$ is due to the quadratic term $\|u(t)\|_U^2$. It may be viewed as a Tykhonov regularization term.

Exercice 2.9. *Prove the uniqueness*

Solution: Let u_1 and u_2 be such that: $j(u_k) = \inf_{v \in M} j(v),\ k = 1, 2.$
We have: $j(tu_1 + (1-t)u_2) < tj(u_1) + (1-t)j(u_2)$.
Hence: $j(tu_1 + (1-t)u_2) < \inf_{v \in M} j(v)$ unless $u_1 = u_2$, which must be the case. □

Remark 2.10. *In the autonomous case (A and B constant), we have:*

$$\|x'(t)\| \leq \|A\|\|x(t)\| + \|B\|\|u(t)\| \leq cst(\|x(t)\|^2 + \|u(t)\|^2)$$

Then, if all hypothesis are satisfied in $I = [0, +\infty$ then $x'(t)$ is in $L^1(I)$ and necessarily the minimizing trajectory $x(t)$ tends to 0 when t tends to $+\infty$.

2.5.2 The Pontryagin principle in the LQ case

(L. Pontryagin, russian mathematician, 1908-1988).

In the case of a non-linear state equation, the cost function is a-priori non-convex and the Pontryagin minimum principle (also called maximum principle) is a necessary condition of optimality.
In the LQ case, the Pontryagin minimum principle is a *necessary and sufficient condition of optimality.* It introduces the Hamiltonian[2], the *adjoint equation*, and the Hamiltonian minimisation/maximisation.

In the present LQ case, the Pontryagin maximum principle reads:

Theorem 2.11. *The trajectory $x(t)$ associated with the control $u(t)$, is optimal for the LQ problem (system (2.1)(2.2)) if there exists an adjoint vector $p(t)$ which satisfies:*

$$p'(t) = -p(t)A(t) + x(t)^T W(t) \text{ for almost } t \in [0, T] \tag{2.4}$$

[2]The so-called "Hamiltonian" in control theory is a particular case of the Hamiltonian in mechanics; it is inspired by the Lagrangian you have studied in (differentiable) optimization.

with the final condition: $p^T(T) = -Qx(T)$. *(p is a line vector).*
Furthermore, the optimal control u satisfies:

$$u(t) = U(t)^{-1}B(t)^T p(t)^T \text{ for almost } t \in [0,T] \tag{2.5}$$

The ideas of the proof based on 'calculus of variations' is similar to those for a PDE model, see Theorem 3.17 in next chapter. Since the proof will be detailed later in the non-linear elliptic PDE case, we refer to [41] for the present proof. □

The theorem above gives an expression of the optimal control depending on the adjoint hence on the state. An explicit expression of the optimal control is given in next section : it is the so-called *feedback law.*

Remark 2.12.
If the term in $x(T)$ *in the cost function is more general and reads* $g(x(T))$, *with* g *any function defined from* $\mathbb{R}^n$ *into* $\mathbb{R}$, C^1 *and convex. That is:*

$$j(u) = \frac{1}{2}\int_0^T \|x(t)\|^2_W dt + \frac{1}{2}\int_0^T \|u(t)\|^2_U dt + g(x(T)) \tag{2.6}$$

Then the final condition of the adjoint reads:

$$\boxed{p(T) = -\nabla g(x(T))} \tag{2.7}$$

Remark 2.13.
a) In case of an infinite time interval ($T = +\infty$), *the final condition becomes:* $\lim_{+\infty} p(t) = 0$.
b) If the system contains a non vanishing source term $s(t)$, *the same result holds.*

2.5.3 The Hamiltonian

(W. Hamilton, 1805- 1865, Irish physicist, astronomer and mathematician).
In this section, the crucial concept of Hamiltonian is introduced.

Let us recall the direct model with no source term ($s(t) = 0$):

$$\boxed{\left\{\begin{array}{l} \text{Given } u(t), \text{ find } x(t) \text{ such that:} \\ x'(t) = A(t)x(t) + B(t)u(t) \text{ for } t \in (0,T) \\ \text{with the initial condition: } x(0) = x_0. \end{array}\right.} \tag{2.8}$$

and the cost function expression:

$$\boxed{j(u) = \frac{1}{2}\int_0^T \|x(t)\|^2_W dt + \frac{1}{2}\int_0^T \|u(t)\|^2_U dt + \frac{1}{2}\|x(T)\|^2_Q} \tag{2.9}$$

In this case the (control) *Hamiltonian* H reads:

$$\boxed{H(x,p,u) = p(Ax+Bu) - \frac{1}{2}(\|x\|_W^2 + \|u\|_U^2)} \tag{2.10}$$

with $H : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$.

In short, the necessary and sufficient condition of the LQ problem solution (Theorem 2.11) reads as: *the Hamiltonian must be maximized.*

Then Theorem 2.11 re-reads as follows: the following *relations* have to be satisfied.

$$\boxed{\left\{\begin{array}{rcll} x'(t) & = & \partial_p H(x,p,u) & = Ax+Bu \\ -p'(t) & = & \partial_x H(x,p,u) & = pA - x^T W \\ 0 & = & \partial_u H(x,p,u) & \text{since } pB - u^T U = 0 \end{array}\right.} \tag{2.11}$$

with the final condition: $\boxed{p(T) = -\nabla g(x(T)) = -x(T)^T Q}$. (Recall, p is a line vector in $\mathbb{R}^n$).

This is the so-called *optimality system.*

The first two equations are the state equation and the adjoint state equations (with the final condition depending on $x(T)$). It is the *Hamiltonian equations.* They are accompanied by the last equation which is the optimality condition on u (necessary and sufficient condition in the present LQ case).
In general these three equations are fully coupled.

The Hamiltonian: the conserved quantity in time.

Indeed, for all t:

$$\begin{array}{rcl} d_t H(x,p,u) & = & \partial_x H(x,p,u)x'(t) + \partial_p H(x,p,u)p'(t) + \partial_u H(x,p,u)u'(t) \\ & = & -p'(t)x'(t) + x'(t)p'(t) + 0 \\ & = & 0 \end{array}$$

Therefore the mapping $t \mapsto H(x,p,u)(t)$ is constant.

In some contexts, the Hamiltonian denotes the energy of the system; in such a case the state and adjoint equations constitute the so-called Hamiltonian equations.
Some links can be done between the Hamiltonian and the Lagrangian (functional introduced in next section). The reader may consult e.g. the on-line course [18] presenting the concepts of "Hamiltonian mechanics" and "Lagrangian mechanics".

2.5.4 Examples

The most simple Linear-Quadratic problem

Let us consider the trivial linear state equation :

$$x'(t) = u(t) \ ; \quad x(0) = x_0 \text{ given.}$$

Given the final time T, we seek to minimize the "quadratic" cost function :

$$j(u) = \int_0^T [-\alpha_1 x^2(t) + \alpha_2 u^2(t)]dt$$

Exercice 2.14.
1) Write the optimality condition.
2) Give an explicit expression of the optimal control.
3) Give an expression of the corresponding optimal trajectory.

The basic car control problem

Let us consider the control of the 1d trajectory of a vehicle (without friction forces). We consider the following idealistic model:

$$x''(t) = -x'(t) + u(t) \ ; \quad x(0) = x'(0) = 0$$

Given the final time T, we seek to maximize the distance travelled with a minimum of energy consumed. This goal can be translated by defining one of the two following cost function to be minimized:

$$j_1(u) = -\alpha_1 x(T) + \alpha_2 \int_0^T u^2(t)dt$$

$$j_2(u) = -\alpha_1 x^2(T) + \alpha_2 \int_0^T u^2(t)dt$$

with α_* given coefficients modeling the weight we want to give to each term.
For a sake of simplicity, we set: $\alpha_1 = \alpha_2 = 1/2$.

Exercice 2.15.
1) Write the optimality condition in both cases: j_1 and j_2.
2) In the case j_1, give the explicit expression of the optimal control .
Next, indicate how to deduce the corresponding optimal trajectory.

2.6 Feedback law and the Riccati equation *

"To go further" section.
Riccati familly (father and son), italian mathematicians, 18th century.

The optimality condition derived above gives an expression of the optimal control $u(t)$ in function of the adjoint solution $p(t)$. In some (simple) cases (e.g. the exercises above), it is possible to integrate explicitly the adjoint equation, resulting to an expression of $u(t)$ depending on the state $x(t)$, that is the so-called *feedback law or closed-loop control*. The latter is the law sough in automatic.

In general, the expression of u in function of x is far to be trivial... But in the LQ case, the feedback law is known: it is obtained by solving the Riccati's equation presented below.

Such feedback laws are not required for VDA; therefore the result below may be skipped for readers interested in VDA only.

2.6.1 Feedback law in the LQ case

In the LQ case we have:

Theorem 2.16. *Under the assumptions of the existence - uniqueness Theorem 2.8, the (unique) optimal control u writes as a feedback function (closed-loop control):*

$$u(t) = K(t)x(t) \text{ with } K(t) = U(t)^{-1}B^T(t)E(t)$$

where the matrix $E(t)$, $E(t) \in M_n(\mathbb{R})$, is solution of the Riccati equation:

$$E'(t) = W(t) - A^T(t)E(t) - E(t)A(t) - E(t)B(t)U^{-1}(t)B^T(t)E(t) \quad \forall t \in (0,T)$$

with the final condition: $E(T) = -Q$.

For all t, the matrix $E(t)$ is symmetric. Furthermore, since Q and W are positive definite, $E(t)$ is positive definite too.

We refer to [41] for the proof. □

This result gives an expression of the optimal control u in function of the state x.

2.6.2 What happens in non-linear cases ?

In practice, optimal control problems are often non-linear, thus the results presented for the LQ problem do not apply directly, in particular the analytic expression of the optimal control. Nevertheless, a good understanding of the LQ solution structure, the strict convexity of the resulting cost function in particular, is useful to tackle non-linear problems.

Then, to solve a *non-linear optimal control problem* (in the sense computing an optimal control u and the optimal trajectory x^u), the option consists to employ *numerical methods* to solve the *necessary first-order optimality* conditions based on the *Hamiltonian*.

***

For Partial Derivatives Equations (PDE) systems (next section), the Pontryagin principle does not apply anymore and the feedback laws are generally not known. These questions are still an active mathematical research field and some recent results exist (eg. the Riccati-like equations for the Navier-Stokes fluid flows equations at low Reynolds number etc).

Nevertheless, for PDE systems, it remains possible to write equations characterizing the optimal control solution: it is the *optimality system* based on the *adjoint equations*.

This approach is developed for non-linear elliptic PDE systems in next chapter.

2.7 The fundamentals equations at a glance

The Linear model

The linear model (without source term $s(t)$) reads:

$$\left\{\begin{array}{l} \text{Given } u(t), \text{ find } x(t) \text{ such that:} \\ x'(t) = A(t)x(t) + B(t)u(t) \text{ for } t \in I = [0,T] \\ \text{with the initial condition: } x(0) = x_0. \end{array}\right. \tag{2.12}$$

The cost function is chosen quadratic:

$$j(u(t)) = \frac{1}{2}\int_0^T \|x^u(t)\|^2_W dt + \frac{1}{2}\int_0^T \|u(t)\|^2_U dt + \frac{1}{2}\|x(T)\|^2_Q \tag{2.13}$$

The model operator $\mathcal{M}(u)$ is defined as:

$$\mathcal{M}: \ u(t) \mapsto x^u(t) \tag{2.14}$$

The operator $\mathcal{M}(u(t))$ is affine in $u(t)$, for all t in $[0,T]$.

The LQ optimal control problem is:
Given x_0 and T, find $u(t)$ such that $x(t)$, solution of the state equation (2.12), minimizes the cost function $j(u)$ defined by (2.13).

The Hamiltonian is the conserved quantity in time. Its expressions is:

$$H(x,p,u) = p(Ax+Bu) - \frac{1}{2}(\|x\|^2_W + \|u\|^2_U) \tag{2.15}$$

with $H: \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, $u(t)$ the control, $x(t)$ the state of the system and $p(t)$ the adjoint state, solution of the adjoint model.
The adjoint model reads:

$$p'(t) = -p(t)A(t) + x(t)^T W(t) \text{ for } t \in [T,0] \tag{2.16}$$

with the final condition: $p^T(T) = -Qx(T)$. (p is a line vector).

The Pontryagin maximum principle states that if the control $u(t)$ is defined as:

$$u(t) = U(t)^{-1}B(t)^T p(t)^T \text{ for almost } t \in [0,T] \tag{2.17}$$

then the state $x^u(t)$ associated to this control $u(t)$, is optimal for the LQ problem.

The Pontryagin maximum principle re-reads: the equations below have to be satisfied.

$$\left\{\begin{array}{rcll} x'(t) & = & \partial_p H(x,p,u) & = Ax + Bu \\ -p'(t) & = & \partial_x H(x,p,u) & = pA - x^T W \\ 0 & = & \partial_u H(x,p,u) & \Longleftrightarrow pB - u^T U = 0 \end{array}\right. \tag{2.18}$$

with the final condition: $p(T) = -\nabla g(x(T)) = -x(T)^T Q$. This is the **optimality system**.

Chapter 3

VDA: optimal control of PDE models, adjoint equations

This chapter presents optimal control basis of PDE systems leading to the Variational Data Assimilation (VDA) formulation. The presentation is focused on the computational aspects: it mainly addresses the derivation of the first order optimality system based on the adjoint equations. Two presentations of how to derive the adjoint equations are proposed: a formal one based on the introduction of the Lagrangian and a more rigorously one. The calculations are derived for a general non-linear elliptic PDE model therefore valid for very wide range of models representing stationary physical phenomena.
The global computational control algorithm (leading to VDA) is highlighted. The algorithm enables the fitting of the model with data. Examples are presented.

Some results and developments require relatively high mathematical skills; this is the case when addressing the differentiability of the PDE solution (with respect to the control variable) and when addressing the existence and uniqueness of the optimal control in the LQ case (PDE version). These mathematical developments are gathered in a dedicated section untitled "mathematical purposes".

Before reading the present chapter, the reader is invited to revise basic but crucial concepts in functional analysis, optimization and gradient-based minimization algorithms. Few of theses basic concepts are shortly recalled in Appendix A. Also, few exercises extracted from the literature are proposed. The reader has to be comfortable with all these basic concepts before going further.

In practice, the numerical resolution of the optimality system enables the *identification of model input parameters* (which are uncertain or unknown) and/or a better *model calibration*.

The outline of this chapter is as follows[1].

Contents

[1]Recall that the sections indicated with a * are "to go further sections". They can be skipped in a first reading or if the reader is not particularly interested in deeper mathematical basis, mathematical proofs.

3.1 The direct model

3.1.1 The general non-linear stationary direct model

Let Ω be a bounded domain (Ω Lipschitz). Let U, a Hilbert space, be the controls space (U Banach space only is potentially enough, but we consider it here Hilbert). Let V, a Hilbert space, be the states space.

We consider the following *state equation* (called *direct model* too):

$$\boxed{\begin{cases} \text{Given } u \in U, \text{ find } y \in V \text{ such that:} \\ A(u;y) = B(u) \text{ in } \Omega \\ \text{with boundary conditions on } \partial\Omega \end{cases}} \tag{3.1}$$

where A is an elliptic operator (with respect to the state y), defined from $U \times V$ into V' (dual of V). $A(\cdot;\cdot)$ is a-priori non-linear, both with respect to the parameter u and with respect to the state y. B is defined from U into V'.

Assumption 3.1. *The state equation (3.1) is well posed in the sense that it has an unique solution $y \in V$, and this solution is continuous with respect to parameters (in particular with respect to u).*

In the linear case (A elliptic operator linear with respect to y) the Lax-Milgram theorem might be the right framework to prove the existence-uniqueness, while the Mint-Browner theorem is useful for a large class of non-linear cases.

Optimal control terminology: distributed control, boundary control

If the control u appears in the "bulk" (i.e. in the equation posed in Ω) then one say it is a *distributed control*.
If u appears on the boundary conditions only, then one say it is a *boundary control*.

3.1.2 Examples

Two simple linear examples based on second order elliptic operators are as follows.

Example 1)

$$A(u;y) = -\lambda \triangle y \text{ and } B(u) = u$$

with mixed boundary conditions: $y = 0$ on Γ_0 ; $-\lambda \partial_n y = \varphi$ on $\partial\Omega/\Gamma_0$, with φ given.
$\lambda \in L^\infty(\Omega)$, $\lambda > 0$ a.e.
The corresponding functional spaces are : $U = L^2(\Omega)$, $V = H^1_{\Gamma_0}(\Omega)$.
The control u is spatially distributed, it is the source term in the Right Hand Side (RHS).
The state equation models a diffusion phenomena (e.g. heat diffusion in a structure, concentration in a fluid, the elastic deformation of a membrane under external force $f = u$, the electrostatic field in a conducting media, etc). Here, u is a distributed control (the external force).

Exercice 3.2.
a) Write the state equation (and recall the adequate functional spaces).
Prove that it has one and only one (weak) solution in V.
b) Prove that the unique solution y is continuous with respect to u
i.e. the operator $\pi : u \in U \mapsto y^u \in V$ is continuous.

Solution.
a) In vertu of Lax-Milgram theorem.
b) The inequalities of continuity and coercivity give the result. □

Example 2)

$$A(u;y) = -div(\lambda \nabla y) \text{ and } B(u) = f$$

with mixed boundary conditions: $y = 0$ on Γ_0 ; $-\lambda \partial_n y = u$ on $\partial\Omega/\Gamma_0$,
with (λ, f) given in the adequate functional spaces.
The correct functional spaces are : $U = L^2(\Omega)$, $V = H^1_{\Gamma_0}(\Omega)$.
In this case u is a boundary control, it represents the flux at boundary.
As previously, the corresponding state equation has we and only solution in V (in vertu of Lax-Milgram theorem), the inequalities of continuity and coercivity give the continuity of y with respect to u.

3.2 The objective and cost functions: misfit between data and the model outputs

If seeking to make fit "at best" (in the least-square sense) the model outputs with the available data z_d, it is natural to measure the discrepancy between both.

The objective function $J(u;y)$ is supposed to be of class C^1.
A natural choice is to define quadratic $J(u;y)$ in the following form:

$$\|y - z_d\|^2_O + \|u\|^2_N$$

The operators O and N are symmetrical positive definite therefore defining norms (potentially semi-definite).
For example, $\|.\|^2_N = (N.,.)_U : \forall v \in U,\ (Nv, v)^2_U \geq c\|v\|^2_U\ ,\ \ c > 0.$

The data (measurements, observations) z^{obs}, are not necessarily the state of the system y. In this case it is necessary to define an *observation operator* C which maps the state of the system y onto the observation space $\mathcal{O}$:

$$\boxed{C : y \in V \mapsto z \in \mathcal{O}} \tag{3.2}$$

($\mathcal{O}$ is supposed to be a Hilbert space).
For complex multi-scale systems, the observation operators may be complex models e.g. modeling 3D fluid dynamic properties from a sequence of 2D satellite images.

Then, we consider the objective function J, $J : U \times V \to \mathbb{R}$, decomposed as follows:

$$\boxed{J(u\,;y) \;=\; J_{obs}(y) \;+\alpha_{reg}\; J_{reg}(u)} \tag{3.3}$$

with J_{obs} the term of misfit with the observations and J_{reg} a regularization term.
The scalar coefficient α_{reg} is a given (positive) weight, balancing the two objective functions terms.

A typical expressions of J_{obs} is as follows:

$$J_{obs}(y) = \Big\| C(y) - z_d \Big\|_O^2 \tag{3.4}$$

where:
- C is the (a-priori non-linear) observation operator,
- z_d a given "data-observation-measurement".

This term measures the discrepancy between a target and the computed solution in the observation space $\mathcal{O}$ (using the metric $\|\cdot\|_O$).

Classical quadratic regularization terms are as follows.

$$J_{reg}(u) = \Big\| u - u_b \Big\|_N^2 \tag{3.5}$$

with u_b is a targeted value of the control u.
This Tykhonov regularization term $\|u - u_b\|_N^2$ penalizes the discrepancy between the control u and the background value u_b, in norm N.
As a consequence, u_b is a (strong) *prior* of the inverse problem...

Another option is to define J_{reg} as follows:

$$J_{reg}(u) = \Big\| D^p u \Big\|^2 \tag{3.6}$$

with D^p (with $p = 1$ or 2 in practice) denoting a differential operator eg the gradient or the Laplace operator.
This term $\|D^p u\|^2$ enforces to find the optimal solution u in a regular (therefore smaller) space eg for $p = 1$, $\|D_p v\|_N^2 = \|\nabla v\|^2$).

Note that we have: $J_{obs} : V \to \mathbb{R}$ and $J_{reg} : U \to \mathbb{R}$.

Next, the cost function to be minimized is defined from the observation function J as follows:

$$\boxed{\begin{array}{c} j(u) = J(u\,;y(u)) \\ \text{where } y(u) \text{ (also denoted } y^u\text{) is the (unique) solution of the direct model (3.1).} \end{array}} \tag{3.7}$$

We have: $j : U \to \mathbb{R}$.

Note that the norm O has to be clarified; it will be defined from a covariance matrix (operator) modeling a-priori knowledge we may have on the solution. If it is simply defined as a diagonal matrix, the diagonal coefficients represent the a-priori confidence we have on each observation. This point is developed in next chapter.

In the following, one need to define the "model operator":

$$\boxed{\begin{array}{c}\mathcal{M} : u \in U \mapsto y^u \in V \\ \text{with } y^u \text{ the (unique) solution of the direct model (3.1).}\end{array}} \tag{3.8}$$

3.3 VDA formulation

Variational Data Assimilation (VDA) denotes the optimal control of a PDE system with the cost function measuring the discrepancy between data and the model outputs.

Let U_{ad}, subset of U, be the admissible control set. Recall the observation functional:

$$J(u\,;y) \;=\; \Big\| C(y) - z_d \Big\|_O^2 \;+ \alpha_{reg} \Big\| u - u_b \Big\|_N^2$$

The optimal control problem reads:

$$\boxed{\left\{\begin{array}{l}\text{Find } u^* \in U_{ad} \text{ such that:} \\ j(u^*) = \min\limits_{U_{ad}} j(u) \\ \quad \text{with the cost function } j \text{ defined by (3.7) .}\end{array}\right.} \tag{3.9}$$

Let us point out again that j depends on the unknown parameter (the control) u through the "model operator" $\mathcal{M}(u) = y^u$; y^u the (unique) solution of the state equation (3.1).

Problem (3.9) can be re-read as:

$$\boxed{\left\{\begin{array}{l}\text{Minimize } j(u) = J(u; y^u) \text{ in } U_{ad} \\ \quad \text{under the "model constraint" (3.1)}\end{array}\right.} \tag{3.10}$$

In other words, the problem is an optimization problem under the constraint "model must be satisfied". This point of view naturally leads to introduce the Lagrangian, see next paragraph.

Exercice 3.3. *Let j be the cost function defined by $j(u) = J(u; y^u)$ with J defined by (3.3).*
a) Write a sufficient condition to have j of class C^1.
b) Write an expression of $j'(u).\delta u$, for all δu
c) Is the cost function $j(u)$ quadratic ? Detail your answer.
Hint: linear + quadratic implies strictly convex.

3.4 On the difficulty to compute the gradient for large size control variables

3.4.1 Global minimization vs local one

To solve the optimization problem (3.10) few approaches are a-priori possible, *depending either if the CPU time T^j_{cpu} required to evaluate the cost function j is reasonable or not...*

If T^j_{cpu} is small (let say in fractions of seconds using your laptop or a super-computer, whatever), then one can adopt a global optimization approach based on stochastic algorithms (e.g. Monte-Carlo), or heuristic methods (e.g. genetic algorithms) or surface response approaches.

If the state equation is a PDE system then generally T^j_{cpu} is not reasonable enough. It may require minutes and even much more...
In this case, global optimization is not worth considering. Then, one adopt *local minimization* approaches based on *algorithms of descent* (see the recall sections in Appendix).

Therefore one need to calculate the expression of the differential $j'(u).\delta u$ for any direction δu, in view to compute its discrete expression i.e. the cost function gradient.

To do so, of course, the functional $j(\cdot)$ must be differentiable, see the previous section.

3.4.2 Relationship between the differential $j'(\cdot)$ and the gradient $\nabla j(\cdot)$

In the computational context, the state equation (direct model) has been discretized using an adequate numerical method (e.g. finite differences, finite elements, finite volumes).

Let us denote U_h the discrete control space with $dim(U_h) = m$. It means there is m discrete control variables.
The gradient $\nabla j(u)$ to be computed, $\nabla j(u) \in \mathbb{R}^m$, is related to the differential $j'(u)$ by the relation:

$$\boxed{< \nabla j(u), \delta u >_{\mathbb{R}^m} = j'(u) \cdot \delta u \text{ for all } \delta u \in U_h \subset \mathbb{R}^m} \tag{3.11}$$

3.4.3 How to compute the cost function gradient ?

The size of the gradient $\nabla j(u)$ equals m. Descent algorithms require scalar products of the form $< \nabla j(u), \delta u >$ for a potentially large number of directions δu; typically m i.e. for each gradient component.

Then, the following question arises:

How to compute the scalar values $< \nabla j(u), \delta u >$ for a large number of directions δu ? (Case m large).

In the continuous formalism, $j'(u) = \frac{dj}{du}(u) \in \mathcal{L}(U;\mathbb{R})$. Then, the question re-reads as:
Given $u_0 \in U_{ad}$ and $\delta u \in U$, how to evaluate efficiently the differential value $\frac{dj}{du}(u_0).\delta u$ for any $\delta u \in U$?

This key question is addressed in a next section with the introduction of the *adjoint model*.

Remark 3.4. *In the case the control variables includes few components e.g. $u = (u_1, u_2)$, then the "gradient" $j'(u) = \frac{dj}{du}(u)$ reads as (recall it is an element of $\mathcal{L}(U;\mathbb{R})$) :*

$$"j'(u) = \frac{dj}{du}(u) = \nabla j(u) = (\frac{\partial j}{\partial u_1}(u), \frac{\partial j}{\partial u_2}(u))^T"$$

and : $j'(u) \cdot \delta u = \frac{\partial j}{\partial u_1}(u) \cdot \delta u_1 + \frac{\partial j}{\partial u_2}(u) \cdot \delta u_2$.

Exercice 3.5. *1) Write the VDA formulation of your practical.*
2) Write an expression of $j'(u).\delta u$

3.5 Mathematical purposes *

This is a "To go further section".

3.5.1 Differentiability of the cost function

In the following, we will need to differentiate the cost function j (with respect to its unique variable u). Thus, the following question is of main interest in order to address the optimal control problem:

Is the cost function (continuously) differentiable ?

This question of differentiability is potentially difficult to answer for non-linear systems. For non-linear hyperbolic system in particular, the solution may be even not continuous with respect to the control variable u...

A useful result to address this question of differentiability is the *implicit function theorem*.

Theorem 3.6. *(Implicit function theorem) Let us assume that:*
i) the operator A and L in the state equation (3.1) are C^1 (i.e. $A \in C^1(U \times V)$ and $B \in C^1(U)$),
ii) the linearized problem is well-posed (i.e. given (u_0, y_0) the linearized operator $\partial_u A(u_0; y_0)$ is

an isomorphism from V into V').

Then, the implicit function $\mathcal{M} : u \mapsto y^u$, with y^u is the (unique) solution of the state equation, is locally C^1 (locally means it exists a neighborhood of u_0 such that).

In short, in view to apply the implicit function theorem we need to verify that the state operators are C^1 and the linearized problem is well-posed.

The implicit function is in fact the "model operator" $\mathcal{M}(u)$, see (3.8).

Example 3) We set: $A(u; y) = -u\triangle y$, $B(u) = f$, with mixed boundary conditions: $y = 0$ on Γ_0 ; $-u\partial_n y = \varphi$ on $\partial\Omega/\Gamma_0$, with φ given.
Here, u is a distributed control, it is the diffusivity coefficient of the material.

Exercice 3.7.
a) Write the corresponding state equation, and prove that it has we and only (weak) solution in V for u given in $L^\infty(\bar{\Omega})$, $u > 0$ a.e..
b) Prove that the unique solution y is continuous and differentiable with respect to u (in the right functional space).

3.5.2 Existence and uniqueness of the optimal control in the LQ case

We refer to [28, 29] for basics on this subject.

LQ problem case

In the Linear-Quadratic (LQ) case, with the existence and uniqueness of the *optimal control* hold.

Theorem 3.8. *Let us assume that the state equation linear and coercive. Let us consider the cost function defined from (3.3)-(3.5), and U_{ad} a closed convex subset of U. It exists a unique solution at the optimal control problem (3.9).*

The state equation is said to be coercive in V if for all $y \in V$, for all $u \in U$, there exists $\alpha > 0$ such that:

$$a(u; y, y) =< A(u; y), y >_{V' \times V} \quad \geq \quad \alpha \|y\|^2$$

Proof. The proof is similar to those of Theorem 2.8.
Let us consider the expression J as in (3.3)-(3.5) with $u_b = 0$.

Step 0) First, let us reformulate the cost function expression as follows:

$$j(u) = \|C(y^u - y^0) + Cy^0 - z_d\|^2 + \|u\|_N^2$$

We set:

$$\pi(u,v) = (C(y^u - y^0), C(y^v - y^0))_Z + (Nu, v)_N \text{ and } B(v) = (z_d - Cy^0, C(y^v - y^0))$$

Since the mapping model $\mathcal{M}$ is affine and continuous, the form π is bilinear symmetric in U and the form L is linear continuous in U.
Furthermore, the form π is coercive in U:

$$\forall u \in U,\ \pi(u,u) \geq c_0 \|u\|^2\ ,\quad c_0 > 0$$

The cost function reads:

$$j(u) = \pi(u,u) - 2L(u) + \|z_d - y^0\|^2$$

Thus j is continuous and satisfies:

$$j(u) \geq c_0 \|u\|^2 - c_1 \|u\| \tag{3.12}$$

A) Proof of existence. It is based on the convergence of minimizing sequence (calculus of variations, D. Hilbert, 1900 a.c. approx.).

Step 1). Let (u_n) be a minimizing sequence:

$$j(u_n) \to_n \inf\{j(u), u \in U_{ad}\} \tag{3.13}$$

From (3.12)(3.13), we obtain:

$$\|u_n\| \leq constant$$

Hence there exists a sub-sequence (u_{n_k}) which converges weakly to a control u in U_{ad}:

$$u_{n_k} \rightharpoonup u \text{ in } U_{ad}$$

Step 2). U_{ad} is a closed convex subset hence *weakly* closed. Hence $u \in U_{ad}$.

Step 3). Since the cost function $j(u)$ is continuous (lower semi-continuous would be enough), we have: $j(u) = \min_{v \in U_{ad}} j(v)$. In other words, u is solution of the optimal control problem.

B) Uniqueness. The bilinear form $v \mapsto \pi(v,v)$ is coercitive hence the cost function is strictly convex. Then, the uniqueness is a straightforward consequence of the strict convexity of $j(u)$. □

Exercice 3.9. *Detail the proof of uniqueness.*
(It is similar to the previous proof for an ODE). □

3.5.3 LQ problems vs real-world problems

Most of the control problems are not LQ problem since the model is generally not linear.

Moreover even if the direct model is linear then one almost never measure the model output everywhere in Ω, that is at the numerical model grid scale ...
Indeed, in real-world problems, data z_d are often available at some locations only, or at larger scale than the model solution scale (eg satellite measurements of local dynamics flows or camera measurements inside a complex multi-scale material).

As a consequence, without imposing a recall to background value u_b that is with $\alpha_{reg} = 0$, the observation functional simplifies as:

$$J(y) \equiv J(u\,;y) \;=\; \Big\| C(y) - z_d \Big\|_O^2$$

In this case, the uniqueness of a global minimum is not garanteed anymore...

One generally have the existence (without always being able to prove it !) but the cost function may be not strictly convex in the whole set U_{ad}.
In practice, the cost function is often "weakly" convex (with "nearly flat valleys"), see eg. Fig. 3.1. This situation leads to an ill-conditioned minimization problem.
The introduction of u_b in J_{reg} acts an excellent "convexification" of J since quadratic; however it is a very strong prior on the sough solution u^*...

More sophisticated regularization treatment than adding a back ground value term $\|u - u_b\|^2$ may be necessary. The current most advanced treatment to this issue is to define "convexifying" metrics, moreover physically-consistent, if one knowns some... This point is briefly addressed in a next chapter.

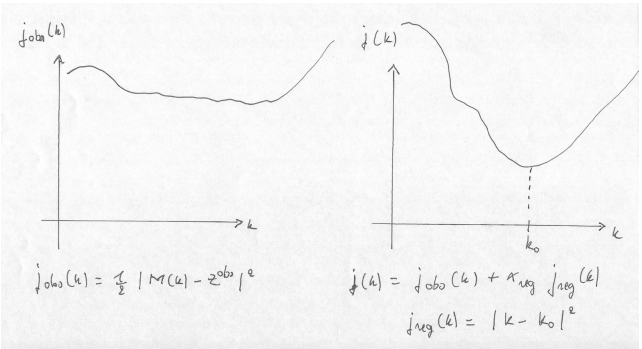


Figure 3.1: Tikhonov regularization. (L) A typical "slightly" convex functional $j_{misfit}(\cdot)$: ill-conditioned minimisation problem. (R) Regularized functional $j(.) = (j_{misfit}(.) + \alpha_{reg} j_{reg}(.))$, with j_{reg} defined from a *prior* value k_0.

3.6 Deriving the optimality system from the Lagrangian

As already mentioned, the optimization problem (3.10) may be viewed as a standard differentiable optimization problem with the model (3.1) being an equality constraint.

3.6.1 Weak forms of the equations

First, let us write the weak (variational) form of the direct model. We set:

$$a(u;y,z) : U \times V \times V \to \mathbb{R} \; ; \quad a(u;y,z) =< A(u;y), z >_{V' \times V}$$

$$b(u;z) : U \times V \to \mathbb{R} \; ; \quad b(u;z) =< B(u), z >_{V' \times V}$$

Let us point out that both $z \mapsto a(.;.,z)$ and $z \mapsto b(.;z)$ are linear (i.e. linear with respect to the test function z).
Furthermore, if the direct model is linear (i.e. with respect to its unknown y) then the form $a(.;y,z)$ is bilinear; it is not if not.

The weak formulation of the *direct model* (the state equation) is as follows:

$$\boxed{\begin{cases} \text{Given } u \in U, \text{ find } y \in V \text{ such that:} \\ a(u;y,z) = b(u;z) \text{ for all } z \in V \end{cases}} \tag{3.14}$$

An advantage to consider the weak form of the equations is to naturally taking into account the boundary conditions in the "single" equation (3.14).
Next, it will be more convenient to derive the correct adjoint equations.

By using the Stampachia representation theorem, (3.14) is equivalent to:

$$A(u;y) = B(u) \text{ in } V' \tag{3.15}$$

3.6.2 The Lagrangian & the optimality system

The optimization problem (3.10) may be viewed as a (differentiable) optimization problem with the model (3.1) being an equality constraint.

Then, it is natural to write the corresponding Lagrangian $\mathcal{L}$:

$$\boxed{\mathcal{L}(u;y,p) = J(u;y) - < A(u;y) - B(u), p >_{V' \times V}}$$

with p the Lagrangian multiplier.
p is a dual variable belonging to the state space V.

Let us calculate the stationary point of the Lagrangian.
The necessary conditions of optimality read:

$$\begin{cases} \partial_u \mathcal{L}(u;y,p).\delta u = 0 & \forall \delta u \in U \\ \partial_y \mathcal{L}(u;y,p).\delta y = 0 & \forall \delta y \in V \\ \partial_p \mathcal{L}(u;y,p).\delta y = 0 & \forall \delta p \in V \end{cases} \tag{3.16}$$

The 1st equation of (3.16) reads:

$$\partial_u J(u;y).\delta u - [\partial_u a(u;y,p) - \partial_u b(u;p)].\delta u \;=\; 0 \quad \forall \delta u \in U$$

It will be shown in next section that this equation is the necessary condition "the gradient must vanish".
Indeed it will be rigorously shown later that:

$$j'(u) = \partial_u J(u;y) - [\partial_u a(u;y,p) - \partial_u b(u;p)] \tag{3.17}$$

The 2nd equation of (3.16) provides the equation:

$$\partial_y a(u;y,p).\delta y \;=\; \partial_y J(u,y).\delta y \quad \forall \delta y \in V$$

The 3rd equation of (3.16) provides the state equation (the direct model):

$$a(u;y,\delta p) \;=\; b(u;\delta p) \quad \forall \delta p \in V$$

It will be shown in next section that the 2nd equation is the so-called adjoint equation.

The equations system above constitutes the so-called "optimality system".

Exercice 3.10. *Re-derive these equations by yourself.*

Remark 3.11. *Note that from the particular decomposition of $J(u;y)$, see (3.3), then one has:*

$$\partial_y J(u,y).\delta y = J'_{obs}(y)\cdot \delta y \text{ and } \partial_u J(u;y).\delta u = J'_{reg}(u)\cdot \delta u \tag{3.18}$$

3.7 Computing the cost function gradient

The goal is the numerical resolution of the optimization problem (3.10). As already mentioned, if the dimension of the discrete control variables u_h is large, this very likely requires to employ descent algorithms therefore to compute the cost function gradient $\nabla j(u_h)$.

3.7.1 The basic Finite Difference-based gradient

The historical method (and the most simple one) consists to approximate the gradient values using finite differences.
Let U_h be the discrete control space, $\dim(U_h) = m$. Then, *given* $\delta u \in \mathbb{R}^m$, the gradient in the direction δu is evaluated as follows:

$$\nabla j(u).\delta u \approx \frac{j(u+\varepsilon\delta u) - j(u)}{\varepsilon} \text{ at order 1 in } \varepsilon \tag{3.19}$$

Then the evaluation of the vector $\nabla j(u)$ requires $(m+1)$ evaluation of j, hence $(m+1)$ resolutions of the direct model...

If T^j_{cpu} is not small and if m not small (typically in geophysical flows, $m \approx 10^6$ or more, see next chapter), then the present finite difference method is not possible.
The finite difference approach is simple to implement, it is a non-intrusive method. Indeed, it is enough to run the direct model as much as required. It is possible if m tiny. On the contrary if m large it is not possible.
Another drawback of this approach is its weak accuracy, depending on the empirical choice of ε.

An other approach would be to solve the linear tangent model but excepted for a very few control parameters, the computational time becomes prohibitive (see next section). Then, the good approach is to solve the adjoint model (see later).

Exercice 3.12. *Propose an algorithm to compute the gradient of your practical by FD.*

3.7.2 The straightforward "gradient" expression

Firstly let us recall the following differential expression :

Lemma 3.13. *Let u_0 in U, for all $\delta u \in U$, we have:*

$$j'(u_0).\delta u \;=\; \frac{\partial J}{\partial u}(u_0; y).\delta u + \frac{\partial J}{\partial y}(u_0; y).w^{\delta u} \tag{3.20}$$

where $w^{\delta u}$ denotes the derivative of the state y with respect to u in the direction δu:

$$w^{\delta u} = \frac{dy}{du}(u_0).\delta u \tag{3.21}$$

Proof. Straightforward calculation (done in exercise). □

3.7.3 The linear tangent model

It is worth noticing that $w^{\delta u} = \frac{dy}{du}(u_0).\delta u$ can be obtained by simply deriving the direct model. This provides the so-called *linear tangent model*.

The linear tangent model derivation

The latter reads as follows:

$$\boxed{\begin{cases} \text{Given } u \in U \text{ and } y^u \text{ the corresponding solution of the state equation (3.1),} \\ \text{ given a direction } \delta u \in U, \text{ find } w \in V \text{ such that:} \\ \dfrac{\partial A}{\partial y}(u; y^u).w = [\dfrac{\partial B}{\partial u}(u) - \dfrac{\partial A}{\partial u}(u; y^u)].\delta u \text{ in } \Omega \\ \text{with corresponding linearized boundary conditions on } \partial\Omega \end{cases}} \tag{3.22}$$

Proof. Straightforward calculation (done in exercise). □

The weak formulation of the *linear tangent model* is as follows:

$$\left\{\begin{array}{l} \text{Given } u \in U \text{ and } y^u \text{ solution of (3.14)},\\ \text{ given } \delta u \in U, \text{ find } w \in V \text{ such that:}\\ \dfrac{\partial a}{\partial y}(u; y^u, z).w = [\dfrac{\partial b}{\partial u}(u; z) - \dfrac{\partial a}{\partial u}(u; y^u, z)].\delta u \text{ for all } z \in V \end{array}\right. \tag{3.23}$$

Solving the linear tangent model provides $w^{\delta u} = \dfrac{dy}{du}(u).\delta u$,
that is the derivative of the state y with respect to the control u in the direction δu.

Recall that: $\mathcal{M} : u \in U \mapsto y^u \in V$. Therefore: $\dfrac{dy}{du}(u) \in \mathcal{L}(U; V)$, and $w^{\delta u} \in V$.

Exercice 3.14. *Write the linear tangent model of your practical; both the weak and the classical forms.* □

Remark 3.15. *It is assumed that the linearized tangent model is well-posed. Let us remark that if we have proved existence of solutions to the non-linear model, one likely had to prove that the linearized model is well-posed.*
Moreover, if the implicit function theorem holds (and has been applied to prove the differentiability of the state with respect to the control), then the linearized problem must be well-posed.
Nevertheless for real-like non-linear problems, even the linearized model analysis can be non straightforward at all...

Resulting sensitivity maps

Let us point out that the linearized tangent model provides $w^{\delta u}$ that is the *local sensitivity of the state y^u with respect to the control u*, at "point" u in the direction δu.
In a modeling context, the resulting sensitivity map (it is distributed values) constitue rich information to better understand the model and/or the modelled phenomena

Exercice 3.16. *In your programming practical, write a formal procedure which plot the "sensitivity map" in this context.* □

FD gradient vs linear tangent model based gradient

The linear tangent model has to be solved m times to obtain $w^{\delta u}$ in each direction δu.
Therefore if m is large and the CPU time for each resolution is large than the linear tangent model approach to compute $w^{\delta u}$ is prohibitive.
However if possible, it is preferable to compute the gradient using the linear tangent model compared to the finite difference method since its accuracy does not depend on the arbitrary setting of ε.

3.8 Rigorous derivation of the adjoint equations

Previously, the optimality system has been formally derived by introducing the Lagrangian and by calculating necessary conditions of stationary points. One of the resulting equation was new; it was the so-called adjoint equation.
Below the adjoint model is rigorously derived.

These equations are a mathematical trick enabling the gradient computation by solving one (1) extra system only; to be compared to $(m+1)$ resolutions if using the finite difference approach or the linear tangent model !

3.8.1 The adjoint model theorem

In this section the expression of the adjoint equations are derived for the general direct model aforementioned.
Let us recall that given F an element of $U' = \mathcal{L}(U, \mathbb{R})$ (U Banach space), we denote indifferently: $F(u) \cdot v \equiv < F(u), v >_{U' \times U} \; \forall v \in U$; $< ., . >_{U' \times U}$ denotes the duality product. The required basic concepts of functional analysis and differential calculus are recalled in Appendix A.

The fundamental result

We have:

Theorem 3.17. *It is assumed that:*
i) the state equation (3.14) is well-posed,
ii) its unique solution y^u is C^1 with respect to u,
iii) the objective function $J(u;y)$ is C^1,
iv) the linearized tangent model (3.23) is well-posed.

Then, the cost function $j(u)$ is C^1, and:

$$j'(u) \;=\; \frac{\partial J}{\partial u}(u; y^u) - [\frac{\partial a}{\partial u}(u; y^u, p^u) - \frac{\partial b}{\partial u}(u; p^u)] \;\; in \; \mathcal{L}(U; \mathbb{R}) \tag{3.24}$$

with $\partial_u J(u; y^u) = \alpha_{reg} J'_{reg}(u)$ if considering the particular decomposition (3.3) of $J(u;y)$.
y^u is the unique solution of the state equation (3.14),
and p^u is solution of the adjoint equation:

$$\left\{ \begin{array}{l} \textit{Given } u \textit{ and } y^u \textit{ the unique solution of (3.14),} \\ \textit{find } p \in V \textit{ satisfying:} \\ \dfrac{\partial a}{\partial y}(u; y^u, p).z = \dfrac{\partial J}{\partial y}(u, y^u).z \qquad \forall z \in V \end{array} \right. \tag{3.25}$$

Its solution p^u (the adjoint state) exists and is unique.
One has $\partial_y J(u; y^u) = J'_{obs}(y)$ if considering the particular decomposition (3.3) of $J(u;y)$.

Proof.

First, under the present assumptions, the implicit function theorem applies and it gives the differentiability of the state with respect to u; i.e. the "model operator" $\mathcal{M} : u \in U \mapsto y^u \in V$ is C^1. Then $j : U \to \mathbb{R}$ can be differentiated. We have: $j'(u) \in \mathcal{L}(U;\mathbb{R})$, and :

$$< j'(u), \delta u >_{U'\times U} \;=\; < \frac{\partial J}{\partial u}(u;y^u), \delta u >_{U'\times U} + < \frac{\partial J}{\partial y}(u;y^u), w^{\delta u} >_{V'\times V} \quad \forall \delta u \in U \tag{3.26}$$

(see Lemma (3.20)) with $< .,. >_{U'\times U}$, $< .,. >_{V'\times V}$ the corresponding duality products.
For sake of clarity we may denote too: $j'(u)\cdot\delta u =< j'(u), \delta u >_{U'\times U}$.

The linear tangent model reads:

$$< \frac{\partial A}{\partial y}(u;y^u).w^{\delta u}, z >_{V'\times V} =< \frac{\partial L}{\partial u}(u).\delta u, z >_{V'\times V} - < \frac{\partial A}{\partial u}(u;y^u).\delta u, z >_{V'\times V} \quad \forall z \in V \tag{3.27}$$

with $w^{\delta u}$ defined by (3.21).
By adding the two equalities above, we obtain:

$$\begin{aligned} < j'(u), \delta u >_{U'\times U} \;=\; & < \frac{\partial J}{\partial u}(u;y^u), \delta u >_{U'\times U} \\ & - < [\frac{\partial A}{\partial u}(u;y^u) - \frac{\partial L}{\partial u}(u)].\delta u, z >_{V'\times V} \\ & + < [\frac{\partial J}{\partial y}(u;y^u) - (\frac{\partial A}{\partial y})^*(u;y^u).z], w^{\delta u} >_{V'\times V} \quad \forall \delta u \in U \; \forall z \in V \end{aligned}$$

where $(\partial_y A)^*$ is the adjoint operator of the linearized direct model operator $\partial_y A$.

In the expression of $j'(u).\delta u$ above, we want to make vanish the term in $w^{\delta u}$.

To do so, we introduce the right equations; it is the so-called adjoint equations.

Let $p^u \in V$ be solution of the problem:

$$< (\frac{\partial A}{\partial y}(u;y^u))^*.p^u, w >_{V'\times V} \;=\; < \frac{\partial J}{\partial y}(u,y^u), w >_{V'\times V} \quad \forall w \in V \tag{3.28}$$

The linearized problem is well-posed, hence the operator $\partial_y A(u;y^u)$ is an isomorphism from V into V', and its adjoint operator $(\partial_y A)^*(u;y^u)$ is an isomorphism from V into V' too, see e.g. [11].

In other respect recall that: $< (\frac{\partial A}{\partial y}(u;y^u))^*.p^u, w >_{V'\times V} \;=\; < p^u, \frac{\partial A}{\partial y}(u;y^u)\cdot w >_{V'\times V}$.

Thus, we obtain the expression:

$$< j'(u), \delta u >_{U'\times U} \;=\; < \frac{\partial J}{\partial u}(u;y^u), \delta u >_{U'\times U} - < [\frac{\partial A}{\partial u}(u;y^u) - \frac{\partial L}{\partial u}(u)].\delta u, p^u >_{V'\times V} \quad \forall \delta u \in U$$

hence the result. □

A few important remarks

- The expression of $j'(u)$ in the direction δu obtained in Theorem 3.17 does not depend anymore on $w^{\delta u}$. Indeed the expression of $j'(u)$ is now explicit with respect to the direction δu.
Thus, if solving the direct model plus the adjoint model then *all components of the gradient* follow i.e. the gradient in all directions !
Let us recall that after discretization (in the finite dimension space U_h), we have:

$$\nabla j(u) \in \mathbb{R}^m, \ \ < \nabla j(u), \delta u >_{\mathbb{R}^m} = j'(u) \cdot \delta u \text{ for all } \delta u \in U_h \subset \mathbb{R}^m$$

- *By construction, the adjoint model is linear*, whatever if the direct model is linear or not. Recall the adjoint is the adjoint operator of the linearized direct operator.

- Let us point out that excepted if the operator A of the state equation is self-adjoint (i.e. $A^* = A$, in other words A linear, symmetric), the adjoint model has a-priori no physical meaning.

- If the direct operator is *self-adjoint*, in other words if $a(u, v)$ is bilinear symmetric then the adjoint operator is the same as the direct operator (but not the right-hand side).
Indeed, in such a case, we have:

$$\partial_y a(u; y^u, p).z = a(u; z, p) = a(u; p, z)$$

Only the source term (RHS) and the boundary conditions differ from the state equation.
Then the differential operator, hence the numerical method and numerical solver, are the same.

Case of non-homogeneous Dirichlet boundary conditions

Let us consider the condition: $y = y_d$ on $\Gamma_d \subset \partial\Omega$. Then, the direct model reads:

$$\begin{cases} \text{Find } y \in V_t = V_0 \oplus y_d \text{ such that :} \\ a(u; y, z) = b(u; z) \text{ for all } z \in V_0 \end{cases}$$

where V_t, affine subspace, is the Dirichlet translation of V_0 (V_0 subspace of V Hilbert).
A typical example for a second order linear elliptic equation is : $V = H^1(\Omega)$, $V_0 = \{z \in V, \ z = 0 \text{ on } \Gamma_d\}$ and $V_t = V_0 \oplus y_d = \{z \in V, \ z = y_d \text{ on } \Gamma_d\}$.

Then the question is : What the non-homogeneous Dirichlet boundary conditions becomes when defining the linear tangent model hence the adjoint model ?

The answer is : *the non-homogeneous Dirichlet condition in the direct model becomes the corresponding homogeneous condition in the linear tangent and adjoint models.*

Let us show this statement in the linear case.
The direct model, if linear in y, re-reads as follows:

$$\begin{cases} \text{Find } y_0 \in V_0 \text{ such that :} \\ a(u; y_0, z) = b(u; z) - a(u; \tilde{y}_d, z) = \tilde{b}(u; z) \text{ for all } z \in V_0 \end{cases}$$

with $y_0 = y - \tilde{y}_d$, $\tilde{y}_d$ being a raising from y_d on Γ_d onto the whole domain Ω. Then, following the proof of Theorem 3.17, it is easy to notice that the corresponding boundary condition in the linear tangent model is homogeneous, hence the same for the adjoint model.

3.8.2 The optimality system

The optimality system denotes the set of equations characterizing the optimal control solution, that is: the state equation, the adjoint state equation and the first order necessary optimality condition.

In the case, $K = U_{ad} = V$ and if considering the particular decomposition (3.3) of $J(u; y)$, the optimality system reads as follows.
The optimal control solution u of Problem (3.9) has to satisfy:

$$\left\{ \begin{array}{l} a(u; y^u, z) \;=\; b(u; z) \quad \forall z \in V \\ a^*((u, y^u); p, z) \;\equiv \partial_y a(u; y^u, p) \cdot z \;=\; J'_{obs}(y^u) \cdot z \quad \forall z \in V \\ j'(u) \cdot \delta u \;=\; 0 = \; \alpha_{reg} J'_{reg}(u) \cdot \delta u - [\partial_u a(u; y^u, p^u) - \partial_u b(u; p^u)] \cdot \delta u \quad \forall \delta u \in U \end{array} \right. \tag{3.29}$$

The optimality system (3.29) is nothing else than the stationary point conditions (3.16) of the Lagrangian. The adjoint state p is the lagrangian multiplier associated to the "-model constraint".

Exercice 3.18. *Write the optimality system which characterizes the solution u of the optimal control problem of your practical. Detail both the weak and the classical forms.*

Remark 3.19. ** ("To go further"). The adjoint equations for a coupled system. If the direct model is composed by two PDEs equations weakly coupled then the adjoint system is composed by the corresponding adjoint equations weakly coupled too by in the reverse way.*
If the direct model is composed by two PDEs equations coupled (fully) then the adjoint system is composed by the corresponding adjoint equations (fully) coupled too.
We do not develop more in detail this section since no time enough.

3.8.3 Gradient components: in weak or classical forms ? *

This a "to go further paragraph".
We have written above the state, linear tangent and adjoint equations in their weak forms for few reasons. First, it is the right framework to study the solutions in the weak sense. Second, deriving the correct adjoint equations in the weak form may be more clear since the weak forms include naturally the boundary conditions. Third, it is the natural framework in the case of the finite element method. Nevertheless, in practice if not considering the finite element method, deriving the equations in their weak forms is not compulsory. Once the reader is confortable with these equation derivations process, it is possible to write all the equations sough directly in the classical forms (or going back from the weak to the classical forms !).

Let us point out that in next chapter a re-derivation of the equations is done in the semi-discrete form (time-dependent problems).

Gradient discretization in the case of FEM In the case of Finite Element discretization, an extra question remains concerning the discretization of the gradient expression (3.24). Recall $j'(u) \in \mathcal{L}(U; \mathbb{R})$. Let us denote $\{\psi_i^u\}_{1 \leq i \leq m}$ the finite element basis of the control variable u. Then the

ith component of the (discretized) gradient is naturally defined as follows :

$$\partial_i j(u) =< j'(u), \psi_i^u >$$

Example Let us consider the direct model : $-\Delta y = u$ (the control is the RHS) with homogeneous Dirichlet boundary conditions on $\partial\Omega$. Let us consider the cost function :

$$j(u) = \frac{1}{2}\int_\Omega (y^u)^2 dx + \frac{1}{2}\int_\Omega u^2 dx$$

If using finite element discretization, then the ith component of the gradient reads :

$$\partial_i j(u) = \int_\Omega p^u \psi_i^u dx + \int_\Omega u\psi_i^u dx \ \ , \ 1 \leq i \leq m$$

with p^u the (discrete) adjoint state function (to be decomposed in its own finite element basis) and u to be decomposed in its finite element basis too.

Finite difference case Now let us consider the same problem but using finite difference schemes (and the same discretization for u and p, with m point values). The ith component of the (discrete) gradient reads :

$$\partial_i j(u) = p_i^u + u_i \ \ , \ 1 \leq i \leq m$$

with p_i^u the ith value of the (discrete) adjoint state.

In the case of *finite element discretization*, there is a choice to make for the gradient definition.
In the example above, the choice would read:

$$\boxed{\int_\Omega p^u \psi_i^u dx \text{ vs } p_i^u}$$

These two possible expressions do not present the same properties, in particular concerning their dependence on the local mesh element size.

3.9 The VDA algorithm

In practice, it is classical to solve the optimization problem (3.9) and the optimality system (3.29) by an iterative descent algorithm, hence a local minimization (on the contrary to global optimization methods).

Given a *first guess* u_0, we seek $(u^m)_m$ which makes decrease the cost function using a first order *descent algorithm* preferably a Quasi-Newton method e.g. BFGS.
The algorithm below is as follows, see Fig. (3.2):

1) compute the cost function using the direct model,

2) compute the gradient using the adjoint model,

3) given the current value u^n, the current values $j(u^n)$ and $\nabla j(u^n)$, compute a new iteration u^{n+1} such that:

$$j(u^{n+1}) < j(u^n)$$

*) Iterate until convergence

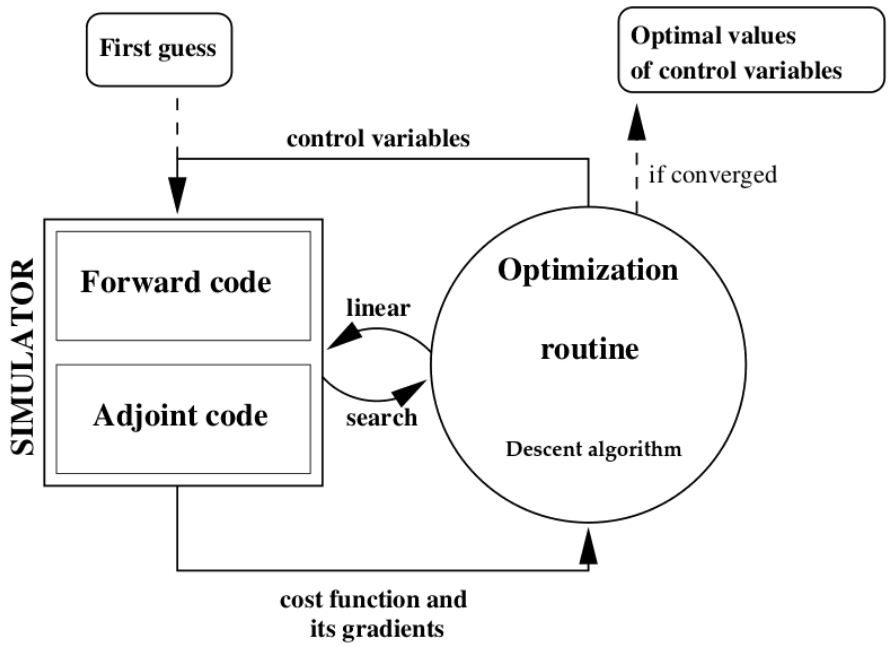


Figure 3.2: VDA algorithm: optimal control of the PDE system (identification process)

Remark 3.20. *In order to solve the optimality system (3.29), the 'natural' method would be use the Newton algorithm. Such a choice is a-priori efficient (quadratic order of convergence) but it requires the computation of the Hessian of j. Generally, its computation is either very complex and/or CPU time consuming, that is why in many cases, we solve the 1st order optimality system (3.29) only.*

The first guess: a potentially important choice The first guess is a first estimate; it is a prior information.
If the cost function is "weakly" convex only, that is presenting kind of "nearly flat valleys", the first guess value may determine the local minimum the descent algorithm will converge to.

Recall that other important priors are the background term (Tykhonov regularization term) and the definition of the norms O and N.
These (semi-)norms may be defined from covariance matrices (symmetric linear positive definite operators) modeling a-priori knowledge we may have on the solution, see later.

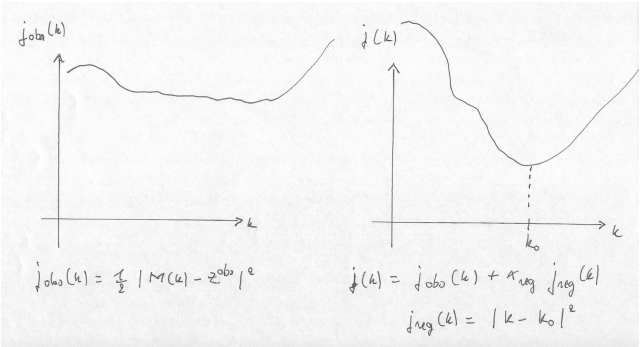


Figure 3.3: The first guess: a potentially important choice

3.10 The fundamental equations at a glance

The non-linear stationary PDE model. The direct model reads:

$$\left\{ \begin{array}{l} \text{Given } u \in U, \text{ find } y \in V \text{ such that:} \\ A(u;y) = B(u) \text{ in } \Omega \\ \text{with boundary conditions on } \partial\Omega \end{array} \right. \tag{3.30}$$

It is supposed to be well-posed.

The model operator $\mathcal{M}(u)$ is defined as: $\mathcal{M}(u) = y^u$.
This operator $\mathcal{M}(\cdot)$ is potentially non linear.

The cost functional $j(u)$ is defined from the "observation" functional $J(u;y)$ as follows:

$$j(u) = J(u\,; y^u) \tag{3.31}$$

where y^u denotes the unique solution of the direct model, given u.

The observation functional $J(u;y)$ is chosen quadratic, decomposed as follows:

$$J(u\,;y) \;=\; J_{obs}(y) \;+\alpha_{reg}\; J_{reg}(u) \tag{3.32}$$

with:

$$J_{obs}(y) \;=\; \Big\| C(y) - z_d \Big\|_O^2 \tag{3.33}$$

$$J_{reg}(u) \;=\; \Big\| u - u_b \Big\|_N^2 \text{ or e.g. } \Big\| D^p u \Big\|_N^2 \text{ with } p = 1 \text{ or } 2. \tag{3.34}$$

The **optimization problem** reads:

$$\left\{ \begin{array}{l} \text{Minimize } j(u) \text{ in } U_{ad} \text{ under the "model constraint"} \\ \text{since } j(u) = J(u; y^u) \text{ with } \; y^u = \mathcal{M}(u). \end{array} \right. \tag{3.35}$$

The adjoint model reads:

$$(\partial_y A)^T(u; y^u) \cdot p \;=\; J'_{obs}(y^u) \text{ in } \Omega \tag{3.36}$$

In weak form:

$$a^*((u, y^u); p, z) \equiv \partial_y a(u; y^u, p) \cdot z \;=\; J'_{obs}(y^u) \cdot z \quad \forall z \in V \tag{3.37}$$

The resulting gradient expression.
The differential of j reads: for all $\delta u \in U$,

$$j'(u) \cdot \delta u \;=\; \alpha_{reg} J'_{reg}(u) \cdot \delta u - [\partial_u A(u; y^u) \cdot \delta u - B'(u) \cdot \delta u] \cdot p^u \tag{3.38}$$

In weak form:

$$j'(u) \cdot \delta u \;=\; \alpha_{reg} J'_{reg}(u) \cdot \delta u - \partial_u a(u; y^u, p^u) \cdot \delta u + \partial_u b(u; p^u) \cdot \delta u \tag{3.39}$$

The resulting gradient $\nabla j(u)$ satisfies:

$$< \nabla j(u), \delta u >_{\mathbb{R}^m} = j'(u) \cdot \delta u \text{ for all } \delta u \in U_h \subset \mathbb{R}^m \tag{3.40}$$

3.11 Another example: control of an elastic membrane deformation *

This is section is a "supplementary material" for those developing the practical programming

Let us consider the deformation of an elastic membrane, fixed at its boundaries, and distorted by an external force f. We denote by d the displacement of the membrane, $d : \Omega \subset \mathbb{R}^N \to \mathbb{R}^N$. It satisfies the following linear second order elliptic system:

$$\left\{ \begin{array}{rcl} -\Delta d & = & f + u \text{ in } \Omega \\ d & = & 0 \text{ on } \partial\Omega \end{array} \right. \tag{3.41}$$

where u is the control of the system, applied on ω a subset of Ω.
The goal is to characterize u such that the displacement d is as close as possible to a target displacement d_0 given. Thus, we define the cost function:

$$j(u) = \int_\Omega |d - d_0|^2 dx + \alpha \int_\omega |u|^2 dx$$

where d is the solution of (3.41) given u. The second term of the cost has two consequences. In a mechanical point of view, it implies that we seek a control as small as possible (in L^2 norm). In a mathematical point of view, it regularizes the cost function: it is the quadratic term essential in the previous existence-uniqueness theorem. In a numerical point of view it "convexifies" the functional; it is what we call a Tikhonov's regularization (see later).

The weight coefficient $\alpha,\ \alpha > 0$ makes a balance between the two terms, it has to be tuned by hand empirically (multi-objective function to be minimized...).
We set $U = (L^2(\Omega))^N$. The set of admissible control K is as follows:

$$U = \{u \in U,\ u = 0 \text{ in } \Omega/\omega\}$$

an the optimal control reads:

$$\min_{u \in K} j(u) \tag{3.42}$$

Existence and uniqueness of solution.
i) Let us point out that in vertu of Lax-Milgram's theorem, given u, the direct model has we and we weak solution d in $V = (H_0^1(\Omega))^N$.
ii) Furthermore, since the mapping $\mathcal{M} : u \in U \mapsto d \in V$ is affine, the cost function j is strongly convex ($j(u) \geq \alpha \|u\|^2_{L^2}$).
In other respect, K is a closed convex set of U. Therefore, in vertu of Theorem 3.8, there exists an unique minimum u to the optimal control problem (3.42).

Exercice 3.21. *Write the optimality system which characterizes the solution u of the optimal control problem (3.42); both in the weak and classical form.*

3.12 How to assess your adjoint computational code & gradient *

We describe below how the adjoint code can be assessed. In short, it is verified that it is actually the adjoint of the tangent linear code by satisfying the scalar product property. Next it is verified that the code computes correctly the partial derivative of the cost function (gradient test) by comparing it with its value obtained by finite differences.
This section may be skipped if not interested by the programming aspects or in his first course reading.

3.12.1 The scalar product test

This test aims at checking if the adjoint code is actually the adjoint of the tangent linear code.
This test supposes to have developed both the adjoint code and the linear tangent code.

This test is equivalent to check the relation (B.1).

- Given an arbitrary $d\mathbf{k} \in \mathcal{K}$, we compute using the tangent linear code :
$$dY = \left(\frac{\partial \mathcal{M}}{\partial \mathbf{k}}\right) \cdot d\mathbf{k}$$
- Given an arbitrary $dY^* \in \mathcal{Y}$, we compute using the adjoint code :
$$d\mathbf{k}^* = \left(\frac{\partial \mathcal{M}}{\partial \mathbf{k}}\right)^* \cdot dY^*$$
- The two following scalar products are computed:
  - $sp_1 = \left\langle dY^*, dY \right\rangle_{\mathcal{Y}}$
  - $sp_2 = \left\langle d\mathbf{k}^*, d\mathbf{k} \right\rangle_{\mathcal{K}}$
- Finally, following the adjoint operator definition $< A\mathbf{k}, Y >=< \mathbf{k}, A^*Y >$ ($A(\cdot)$ linear), we check that: $sp_1 = sp_2$.

Figure 3.4 (b) shows a typical example of the scalar product test.

```
####################################
##    TEST DU PRODUIT SCALAIRE    ##
####################################

Appel du code linaire tangent...
Appel du code adjoint...
<Xd,Xb> =   -682.277083033688
<Yd,Yb> =   -682.277082428555
relative error :  -8.869324203484993E-010
```

Figure 3.4: Adjoint code validation: scalar product test

3.12.2 The gradient test

The objective of this test is to check if the adjoint variables dX^* computed by the adjoint code actually correspond to the partial derivatives of the cost function. This is a crucial test.
This test requires to perform the direct code a few times and the adjoint code.

The Taylor expansion of the cost function j at $\mathbf{k}$ for a small perturbation $\alpha\,\delta\mathbf{k}$ (where $\alpha \in \mathbb{R}^+$) reads:

$$j(\mathbf{k}+\alpha\,\delta\mathbf{k}) \;=\; j(\mathbf{k}) + \alpha\,\frac{\partial j}{\partial \mathbf{k}}(\mathbf{k})\cdot\delta\mathbf{k} + o\left(\alpha\|\delta\mathbf{k}\|\right)\,. \tag{3.43}$$

It follows the uncentered finite difference approximation (order 1) and the centered finite difference approximation order 2:

$$\frac{j(\mathbf{k}+\alpha\,\delta\mathbf{k}) - j(\mathbf{k}-\alpha\,\delta\mathbf{k})}{2\alpha} \;= \frac{\partial j}{\partial \mathbf{k}}(\mathbf{k}).\delta\mathbf{k} + O\left(\alpha^2\|\delta\mathbf{k}\|^2\right)\,. \tag{3.44}$$

Then, we set either

$$I_\alpha \;=\; \frac{j(\mathbf{k}+\alpha\,\delta\mathbf{k}) - j(\mathbf{k}-\alpha\,\delta\mathbf{k})}{2\alpha\,\frac{\partial j}{\partial \mathbf{k}}(\mathbf{k})\cdot\delta\mathbf{k}}\,. \tag{3.45}$$

or:

$$I_\alpha \;=\; \frac{j(\mathbf{k}+\alpha\,\delta\mathbf{k}) - j(\mathbf{k})}{\alpha\,\frac{\partial j}{\partial \mathbf{k}}(\mathbf{k})\cdot\delta\mathbf{k}}\,. \tag{3.46}$$

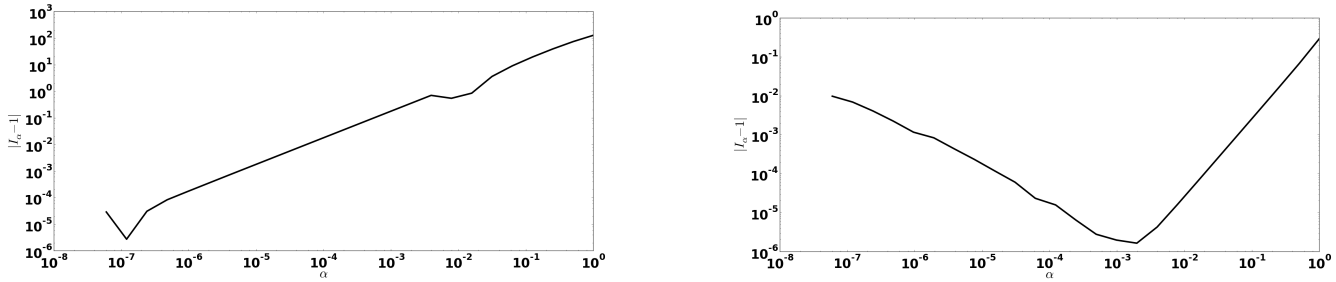
According to (3.43), one must have: $\lim\limits_{\alpha\to 0} I_\alpha = 1$.

The gradient test consists to check this property as follows.

- For an arbitrary $\mathbf{k}$, compute $\frac{\partial j}{\partial \mathbf{k}}(\mathbf{k})$ with the adjoint code.
- Using the direct code, compute $j(\mathbf{k})$.
- For $n = 0, \ldots, N$:
  - Compute $\alpha_n = 2^{-n}$;
  - Using the direct code, compute $j(\mathbf{k}+\alpha_n\,\delta\mathbf{k})$;
  - Compute I_{α_n} ;
- Check if $\lim\limits_{\alpha\to 0} I_{\alpha_n} = 1$ or not.

Figure 3.5 shows two results of the gradient test: at order 2 and at order 1. $|I_\alpha - 1|$ is plotted vs α in logarithmic scale.
The convergence is good until $\alpha > 10^{-7}$.
However, observe the difference of accuracy between the 1st order and 2nd order approximation.

If α is smaller than $\approx 10^{-7}$, the approximation of the partial derivatives is not reliable anymore due to truncation errors.
(To show this, one can add a fix term in the Taylor expansion and notice that it is divided by the perturbation therefore increasing).



(a) (b)

Figure 3.5: The adjoint code validation. Gradient test at order 1 (a), at order 2 (b).

Exercice 3.22. *Perform the gradient test for your practical problem.*
Is your gradient computation valid ?

Chapter 4

Equivalences between VDA, the BLUE and Bayesian inference

If the forward model is linear, if the observation operator C is linear and if the functional $J(u;y)$ is quadratic (in its primal variables) that is *if the considered inverse problem is the LQ problem* then the Best Linear Unbiased Estimator (BLUE) is equivalent to a particular VDA solution u^*.
This result is demonstrated in the first section.

Moreover, if assuming Gaussian probability density functions (pdf) then the same VDA solution is nothing else than the most probable solution, that is the Maximum A-Posteriori (MAP) estimation provided by the Bayes theorem.
This result is demonstrated in the third section.

In the second section, it is shown how the introduction of covariance operators may improve the robustness and the efficiency of the optimisation process by defining "consistent-model" metrics.
These covariances operators (matrices in discrete version) may be interpreted as prior probability model of the studied phenomena.

The outline of this chapter is as follows.

Contents

4.1 The Best Linear Unbiased Estimator (BLUE) and VDA

The basic VDA approach which has been presented in the previous chapter is purely deterministic. It provided the optimal control denoted by u^*, solution of (3.9). Below it is shown that the Best Linear Unbiased Estimator (BLUE) i.e. the most simple statistic estimator provides a particular VDA solution. This result suggests to define the norms (metrics) measuring the discrepancy between the model output and data as the inverse of the covariance matrix of errors.

4.1.1 Least-square solutions depend on the metric(s)

Let us consider the following simple but instructive example. One consider two measurements of a scalar quantity u: $u_1 = 1$ and $u_2 = 2$.
'Naturally', one would seek u minimizing the following cost function:

$$j(u) = (u - z_1)^2 + (u - z_2)^2$$

It is a "explicit" (standard) least-square problem since u does not have to satisfy an underlying model. The solution is: $u^* = \frac{3}{2}$.

Now, let us assume that the second data had measured the quantity $2u$ and not u. Then, the two measurements are: $z_1 = 1$ and $z_2 = 4$.
Again, 'naturally', one seek u minimizing the following cost function:

$$g(u) = (u - z_1)^2 + (2u - z_2)^2$$

The solution is: $u^* = \frac{9}{5}$. This solution differs from the first one !

This simple example illustrates that the solution of the minimization problem (here a trivial linear least-square problem) depends on the considered metrics (actually norms) employed in the cost function.
The standard least-square solution is dependent of the measurement metric or accuracy !

In presence of errors (which is always the case in real-life problems), it does not take into account the confidence we may have on each measurement.
In order to take into account the measurement accuracies, one can *introduce a-priori information in the norm* measuring the discrepancy.

4.1.2 The BLUE, scalar case

We refer the reader to Appendix A.4 for very basic recalls in statistics and the employed notations. BLUE means Best Linear Unbiased Estimator.

Definition 4.1. *The Best Linear Unbiased Estimator (BLUE) of an aleatory variable $\hat{X}$ based on data Z is as follows:*

a) *It a linear function of Z.*
(Comment: it is the most simple).

b) *It is unbiased: $E(\hat{U}) = u$.*
(Comment: it is desirable).

c) *It has the smallest variance $Var(\hat{X})$ among all unbiased linear estimation.*
(Comment: optimal accuracy is expected).

BLUE for the previous basic example Let us denote the two measurements including errors as follows:

$$z_i = u_i + \varepsilon_i, \ i = 1, 2$$

The errors of measurements ε_i are supposed to be:
- unbiased: $E(\varepsilon_i) = 0$.
(Comment: sensors are unbiased; mean of errors vanishes).
- with a variance given: $Var(\varepsilon_i) = \sigma_i, \ i = 1, 2$.
(Comment: the accuracy of each sensor is known).
- uncorrelated : $E(\varepsilon_1 \varepsilon_2) = 0$.
(Comment. Measurements are independent hence the covariance vanishes; in addition, means vanish hence the relation).

Let us calculate the Best Linear Unbiased estimation (BLUE) denoted by u^*. (Here "best" means giving the lowest mean-square error).

By definition, the BLUE satisfies: $u^* = a_1 z_1 + a_2 z_2$ (linear relationship).
The coefficients a_i have to be determined.
The errors are unbiased:

$$E(u^* - u) = 0$$

We have:

$$E(u^*) = (a_1 + a_2)u + a_1 E(\varepsilon_1) + a_2 E(\varepsilon_2) = (a_1 + a_2)u$$

Therefore: $a_2 = (1 - a_1)$.

Next, let us calculate the variance of u^*:

$$\begin{aligned} Var(u^*) &= E((u^* - u)^2) = E((a_1 z_1 + a_2 z_2)^2) \\ &= a_1^2 E(\varepsilon_1^2) + 2a_1 a_2 E(\varepsilon_1 \varepsilon_2) + a_2^2 E(\varepsilon_2^2) \\ &= a_1^2 \sigma_1^2 + a_2^2 \sigma_2^2 \\ &= a_1^2 \sigma_1^2 + (1 - a_1)^2 \sigma_2^2 \end{aligned}$$

By definition, the BLUE u^* minimize the variance. The latter is minimal if its derivative with respect to a_1 vanishes; this reads: $a_1 = \frac{\sigma_2^2}{\sigma_1^2 + \sigma_2^2}$.

Therefore, the BLUE reads:

$$\boxed{u^* = \frac{1}{(\frac{1}{\sigma_1^2}+\frac{1}{\sigma_2^2})}(\frac{1}{\sigma_1^2}z_1 + \frac{1}{\sigma_2^2}z_2)} \tag{4.1}$$

Moreover: $Var(u^*) = \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2+\sigma_2^2}$. Therefore:

$$\frac{1}{Var(u^*)} = \frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}$$

4.1.3 Equivalence with a VDA solution

It is straightforward to verify that the BLUE u^* defined by (4.1) minimizes the following quadratic cost function:

$$\boxed{j(u) = \frac{1}{2}\frac{1}{\sigma_1^2}(u-z_1)^2 + \frac{1}{2}\frac{1}{\sigma_2^2}(u-z_2)^2} \tag{4.2}$$

We have:

$$j''(u) = \frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2} = \frac{1}{Var(u^*)}$$

Therefore, the Hessian of j (defining the "convexity rate" of j) equals the estimation accuracy.

This property between the Hessian and the analysis quality can be simply illustrated in 1D, see Fig. 4.1.

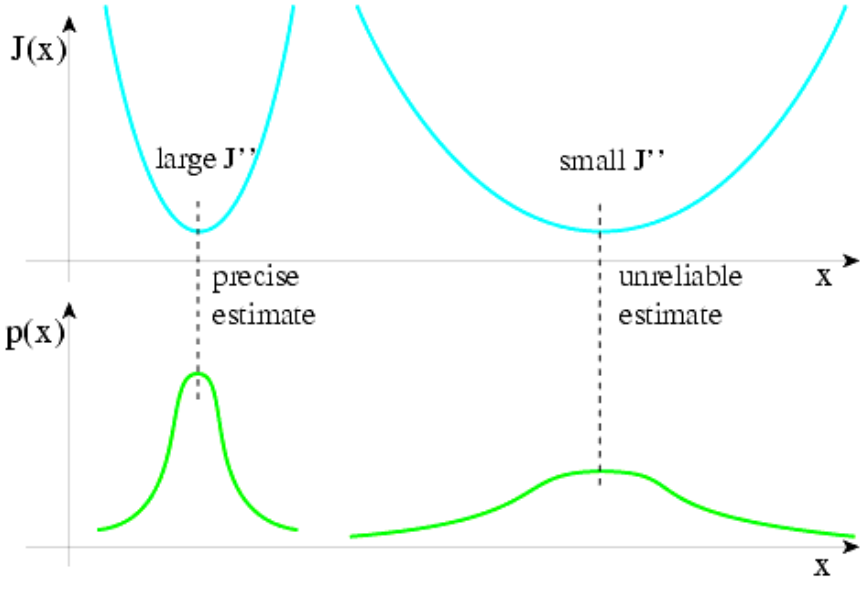


Figure 4.1: In the LQ case: convexity=reliability. In 1D, the Hessian=second derivative hence the "convexity rate". The latter measures the pdf sharpness of the analysis. Figure extracted from F. Bouttier, P. Courtier course, ECMWF (www.ecmwf.int).

Let us remark that the BLUE calculated above (with unbiased measurements) minimizes the following cost function too:

$$j(u) = \frac{1}{2}(u - z_1, u - z_2) \begin{pmatrix} \sigma_1^2 & \rho_{12}\sigma_1\sigma_2 \\ \rho_{12}\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix}^{-1} \begin{pmatrix} u - z_1 \\ u - z_2 \end{pmatrix} = \frac{1}{2}\|u - z\|_N^2 \tag{4.3}$$

In this cost function expression, the norm N considers that the observations errors are correlated. Indeed the extra diagonal terms are non vanishing.

Extension to m observations The extension to m observations is straightforward. We denote the m measurements with errors by: $z_i = u + \varepsilon_i,\ i = 1, .., m$. We assume that the errors are:
- unbiased: $E(\varepsilon_i) = 0,\ i = 1, .., m$.
- with a variance given: $Var(\varepsilon_i) = \sigma_i,\ i = 1, .., m$.
- uncorrelated : $E(\varepsilon_i \varepsilon_j) = 0$ for all $(i, j),\ 1 \leq i, j \leq m,\ i \neq j$.

Then, the Best Linear Unbiased Estimation reads:

$$u^* = \frac{1}{\sum_{i=1}^m \frac{1}{\sigma_i^2}} (\sum_{i=1}^m \frac{1}{\sigma_i^2} z_i) \text{ with } \frac{1}{Var(u^*)} = \sum_{i=1}^m \frac{1}{\sigma_i^2}$$

Therefore the BLUE minimizes:

$$\boxed{j(u) = \frac{1}{2} \sum_{i=1}^m \frac{1}{\sigma_i^2} (u - z_i)^2} \tag{4.4}$$

Again, the Hessian (here a scalar second derivative) defines the "convexity rate" and mesures the "sharpness" of the analysis:

$$\boxed{j''(u) = \frac{1}{Var(u^*)}}$$

A second simple and illustrative example of BLUE and VDA solution is proposed in Appendix.

4.1.4 Sequential filters formalism

Filters are stochastic algorithms which operate recursively on streams of noisy input data to produce a statistically optimal estimation of the underlying state.

The scalar case Let go back to the simple example with two observations. In this case, the BLUE reads:

$$u^* = \frac{\sigma_2^2 z_1 + \sigma_1^2 z_2}{\sigma_1^2 + \sigma_2^2} = z_1 + \left(\frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2} \right) (z_2 - z_1)$$

Let us consider that:
a) z_1 is a first estimation. It is the so-called *background* or *first guess* value.
b) z_2 is an independent observation.

We denote the first-guess value z_1 by u_b : $u_b \equiv z_1$.
We denote the next observation z_2 by z : $z \equiv z_2$.
Following this point of view, the BLUE u^* reads :

$$\boxed{u^* = u_b + (\frac{\sigma_b^2}{\sigma_b^2 + \sigma_0^2})(z - u_b)} \tag{4.5}$$

Using the terminology of sequential data assimilation (e.g. the Kalman filter), this equation reads as follows:
"The best estimation u^* equals the first guess + the *gain* times the *innovation* $(z - u_b)$".

The gain is defined as the multiplicative factor of the innovation.
In a filter algorithm (eg. the Kalman filter), defining the gain matrix is the key point.

On the Kalman filter. *R. E. Kalman, Hungarian-born American electrical engineer, born in 1930.*

In short, the Kalman filter is not affordable for CPU-time consuming models (despite the "easy" quadratic problem solved). Moreover it is not optimal for nonlinear models.
For non-linear models and/or expensive models, one has to consider Ensemble Kalman filters (e.g. EnKF, ETKF), or ... VDA ("4D-Var") .
Concerning sequential data assimilation methods (e.g. the Kalman filter), the reader may consult e.g. [19].

From Wikipedia's page: "Kalman filtering, also known as Linear Quadratic Estimation (LQE), works in a two-step process. In the prediction step, the Kalman filter produces estimations of the current state variables, along with their uncertainties. Once the outcome of the next measurement (necessarily corrupted with some amount of error, including random noise) is observed, these estimations are updated using a weighted average, with more weight being given to estimations with higher certainty. Because of the algorithm's recursive nature, it can run in real time using only the present input measurements and the previously calculated state and its uncertainty matrix; no additional past information is required".

4.1.5 The BLUE (vectorial case) & resulting optimal metric

The goal of this section is to calculate the BLUE in a vectorial case; then, deduce the required norm in the VDA cost function to obtain the BLUE.

Let us consider the previous basic example but with n variables and m observations.
We seek to estimate the *vectorial* quantity $u = (u_1...u_n)^T \in \mathbb{R}^n$.
The measurements-observations are: $z^{obs} = (z_1^{obs}...z_m^{obs})^T \in \mathbb{R}^m$.
We assume that the *observation operator* C is linear: C is a $m \times n$ matrix.

We consider errors e on the observations:

$$z^{obs} = Cu + e \qquad x \in \mathbb{R}^n,\ z \in \mathbb{R}^m \tag{4.6}$$

e, $e \in \mathbb{R}^m$, denotes the observation-modelling error.

This error e is supposed to be an aleatory variable which satisfies:

1) $E(e) = 0$ i.e. the sensors and/or model are unbiased.
(This assumption may be reasonable (or not...) in real-world problems).

2) Its covariance matrix $Cov(e)$, $Cov(e) = E(ee^T)$ is given.
That is the covariances of observations-modeling errors are supposed to be known.

This last assumption is generally not satisfied in real-world problems; however as shown below it is the adequate matrix to consider...

If the observations-modeling errors are not independent then the matrix $Cov(e)$ is non-diagonal.
Recall that each diagonal term of $Cov(e)$ represents the confidence - accuracy one may has on the measurement-modeling error.

Under these assumptions and by using the *Gauss-Markov theorem*, the following result can be proved.

Proposition 4.2. *The BLUE is the (unique) solution of the minimization problem* $\min_x j(u)$ *with:*

$$j(u) = \frac{1}{2}\|Cu - z^{obs}\|^2_O \text{ with } O = (Cov(e))^{-1}. \tag{4.7}$$

Recall the present discrete estimation problem would correspond to a linear direct model with a linear observation operator and errors on the observations.

Then this result demonstrates that *if the exact covariances of the observation-modelling errors* $Cov(e)$ *are known* (this is unfortunately not unrealistic) *then the matrix to consider in the metric of* $j(u)$ *to obtain the BLUE is* O *with* $O = (Cov(e))^{-1}$.

The reader can find more details in one of the numerous excellent books treating of the subject, see e.g. [19].

Exercice 4.3. *Write the linear observation operator* C *in the following case:*
Two observations z_1, z_2 *of* u_1 *are available, while one observation* z_3 *of* u_2 *only is available.*

4.2 The Bayesian view

Rev. Thomas Bayes (1702-1761), English statistician and philosopher. In probability and statistics, Bayes' theorem describes the probability of an event, based on prior knowledge of conditions that might be related to the event. Independently of Bayes, Pierre-Simon Laplace in 1774 used conditional probability to formulate the relation of an updated posterior probability from a prior probability, given evidence. Source: Wikipedia.

One of the many applications of Bayes' theorem is Bayesian inference, a particular approach to statistical inference.

The Baye's law. Let u be the uncertain parameter to be estimated and z^{obs} an observation.
Let $p(u)$ be the prior distribution of u.
$p(u|z^{obs})$ (the probability of z^{obs} given u) is the *posterior distribution.*
The Baye's theorem states that the posterior distribution satisfies:

$$p(u|z^{obs}) = \frac{p(z^{obs}|u)}{p(z^{obs})} p(u) \tag{4.8}$$

where $p(z^{obs}|u)$ is the *likelihood function* for u.

Note that one can obtain $p(z^{obs})$ by integrating over all u: $p(z^{obs}) = \int p(z^{obs}|u)\ p(u)dx$.

Seeking the most probable parameter u. Let us seek *the most probable parameter* u, given the observations z^{obs} (and potentially given the background u_b too):

$$\boxed{u^* = arg\max_u \Big(p(u|z^{obs} \text{ and } u_b)\Big)} \tag{4.9}$$

This may be done by defining the following cost function:

$$j(u) = -\log(p(u|z^{obs} \text{ and } u_b)) + c \tag{4.10}$$

for any constant c. Indeed, log is a monotonic function then it follows:

$$u^* = arg\min_x j(u) \tag{4.11}$$

Assuming that the observation errors and the background errors are uncorrelated (this seems highly reasonable):

$$p(z^{obs} \text{ and } u_b|u) = p(z^{obs} \text{ and } u_b)p(u)$$

For a sake of simplicity, from now we skip the background variable u_b.

Using the Bayes law (4.8), it follows:

$$j(u) = -\log p(z^{obs}|u) - \log p(u) + \text{ c} \tag{4.12}$$

for any constant c.

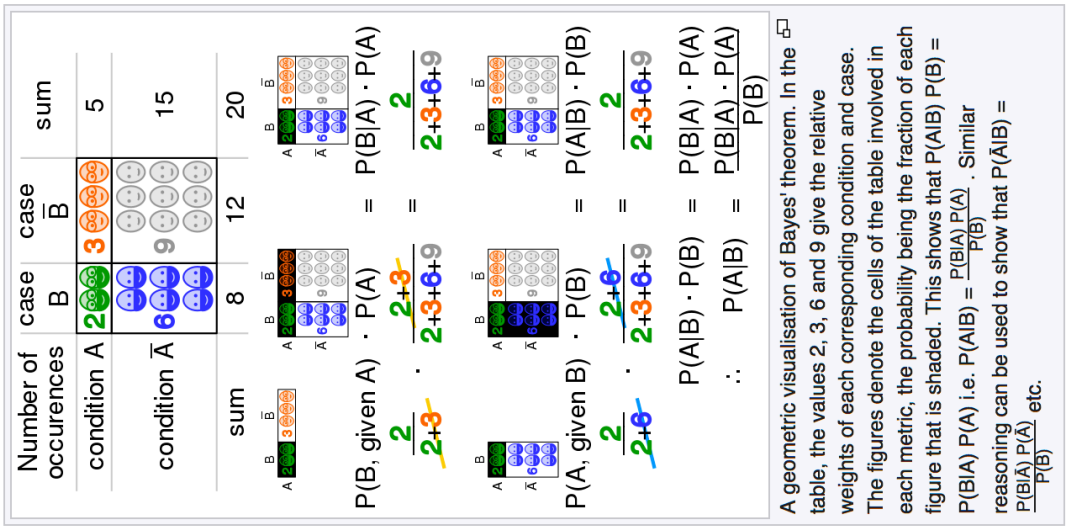


Figure 4.2: The Bayes theorem for discrete events. Figure extracted from Wikipedia.

If assuming Gaussian pdf... The maximum likelihood approach is applicable to any pdf (probability density function) $p(z^{obs}|u)$ and $p(u)$. However, let us consider the special case of Gaussian pdf. Note that the chosen log function in the definition of j above is particularly well-adapted to a Gaussian function ...

- We assume that the prior distribution $p(u)$ is Gaussian. More precisely

$$p(u) \sim \mathcal{N}(u_b, \sigma_b^2)$$

with: the mean u_b, the variance σ_b^2, the standard deviation $\sigma_b = \|B\|$, B the background error covariance matrix defined as previously suggested:

$$B = Cov(e_b) = E(e_b e_b^T) \tag{4.13}$$

We have:

$$p(u) = \frac{1}{(2\pi\sigma_b)^{1/2}} \left(-\frac{1}{2}\|u - u_b\|^2_{B^{-1}}\right)$$

- We assume the likelihood $p(z^{obs}|u)$ to be Gaussian too.
Note that this assumption is a-priori not verified in particular for non-linear models....
We set:

$$p(z^{obs}|u) \sim \mathcal{N}(u_o, \sigma_o^2)$$

with : mean z^{obs}, variance σ_o^2, standard deviation $\sigma_o = \|O\|$, O the observation error covariance matrix defined by:

$$O = Cov(e_o) = E(e_o e_o^T) \tag{4.14}$$

We have:

$$p(z^{obs}|u) = \frac{1}{(2\pi\sigma_o)^{1/2}} \exp\left(-\frac{1}{2}\|Cu - z^{obs}\|^2_{O^{-1}}\right) \tag{4.15}$$

By taking the log in (4.12) (and choosing an adequate constant value), it follows:

$$\boxed{j(u) = \frac{1}{2}\|Cu - z^{obs}\|^2_{O^{-1}} + \frac{1}{2}\|u - u_b\|^2_{B^{-1}}} \tag{4.16}$$

with the norms O and B defined as presented above.

In summary, computing the most probable posterior value involves maximising a product; next taking the log of the posterior pdf in the Gaussian case leads to the minimization of the "usual" VDA formulation for the LQ case (quadratic terms) with metrics defined from the error covariance matrices.

Limitations of this analysis The equivalency shown above assumes a Gaussian prior distribution of u (this may be reasonable) and a Gaussian likelihood $p(z^{obs}|u)$.
However, this last assumption is a-priori not satisfied for non-linear problems...
And for an a-priori Gaussian likelihood function $p(z^{obs}|u)$, the equivalent cost function $j(u)$ is quadratic.
This is a-priori not true for $p(z^{obs}|u)$ non Gaussian...

The resulting MAP estimation Following the Bayes theorem, see (4.8), the posterior distribution is a Gaussian distribution which reads (see eg. Lorenc's course):

$$p(u|z^{obs}) = \frac{1}{(2\pi\sigma_p)^{1/2}} \exp\left(-\frac{1}{2}\|u - u_p\|^2_{P^{-1}}\right) \tag{4.17}$$

with: the mean u_p, the standard deviation $\sigma_p = \|P\|$, P satisfying:

$$P^{-1} = B^{-1} + C^T O^{-1} C \tag{4.18}$$

And

$$u_p = u_b + PC^T O^{-1}(z^{obs} - Cu_b) \tag{4.19}$$

That is the *Maximum A-Posteriori (MAP) estimation.*

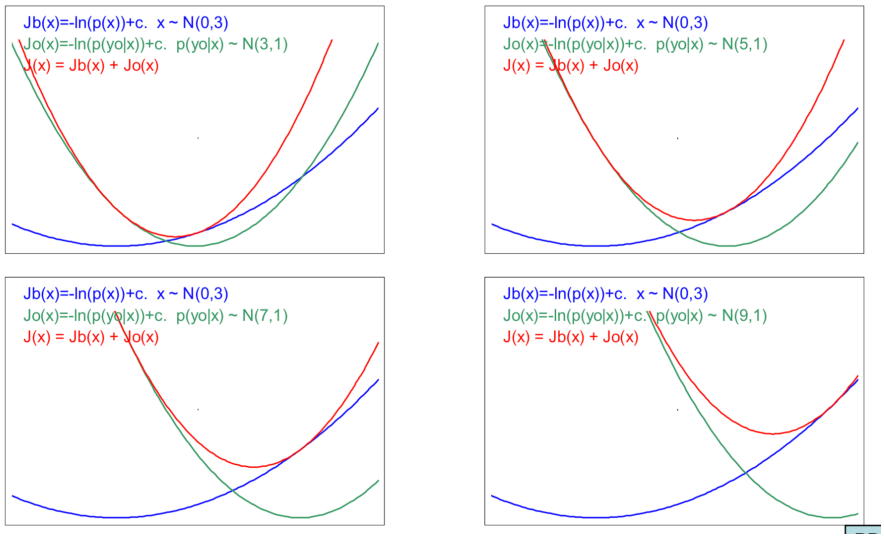


Figure 4.3: Figure extracted from A. Lorenc course, Met Office (JCSDA Summer School, 2009).

4.3 Conclusion: in the LQ case, VDA, the Maximum A-Posteriori (MAP) estimation and BLUE can be equivalent

In this section, it has been shown that in the LQ case (the model is linear, the observation operator is linear therefore the cost function is strictly convex and it exists an unique minimum):

. The VDA solution equals the BLUE if the exact error covariances are known and define the employed metric in $j(u)$.

. The Hessian ("convexity rate") of $j(u)$, D^2j, may be interpreted as the estimation accuracy. With the previous notations, one has:

$$D^2j = C^T O^{-1} C + B^{-1} = [Cov(C(u^{true}) - z_d)]^{-1} \tag{4.20}$$

Recall that D^2j is $m \times n$ matrix.

. If assuming Gaussian pdf, the VDA solution based (on the metrics above) equals the most probable solution.
It is the Maximum A-Posteriori (MAP) estimation provided by the Bayes theorem.

In summary, in the LQ case and if considering the error covariance matrices as metrics, all the equations - approach provide the same solution (optimal paramter) !

What about in non-linear cases ?

If the direct model is linear and the observation operator is linear than the cost function is strictly convex (LQ problem); it admits a unique (global) minimum.
On the contrary, if the direct model is non-linear and/or the observation operator is non-linear than the cost function is a-priori non convex.
In such a case, the minimization process may converge to a local minimum, depending on the priors.
The minimization problem may be more or less well conditioned depending on the Hessian properties of j (its "convexity rate")... See eg. Fig. 3.1.
Moreover in such a case, the equivalence between a particular VDA solution and the BLUE does not hold anymore; the Gaussian assumption in the Bayes framework does not hold anymore too.
However, the analysis for the LQ case show what optimal metric, norms or covariance operators should be chosen.
This suggest to consider "model-consistent" norms (the covariance operators) in the definition of $j(\cdot)$...

Filters in non-linear cases. Filtering methods can be extended to non-linear systems for example by linearizing the model at each time step; this the basic idea of the Extended Kalman's Filter. However in this case, the filter is not optimal anymore. Many others extensions exist. The Ensemble Kalman filter is the most common one. Briefly, it consists to perform a Monte-Carlo algorithm to estimate the covariance errors matrices. Others filters have been developed in particular to reduce the required CPU time to compute the gain matrices, see e.g. the SEEK filter. On the filters approaches, the reader may consult e.g. [19] .

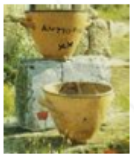
4.4 Another example of BLUE and VDA solution

These slides have been extracted from Prof. Olivier Thual's course (University of Toulouse, Fluid Mech. Institute).

The Best Linear Unbiased Estimate (BLUE) in image and for a simple example: What Time Is It ?

An example due to Prof. O. Thual
Univ. Toulouse INPT & CERFACS
See O. Thual webpage





Horloges de précisions égales

- Horloge 1 : il est 14h23mn à 5mn près
- Horloge 2 : il est 14h19mn à 5mn près

The BLUE is: $14H21\text{mn}$ at $+/-$ $5\sqrt{2}$ mn$(\approx 3,5$ mn$)$
ie. the mean at +/- the variance

Horloges de précisions inégales

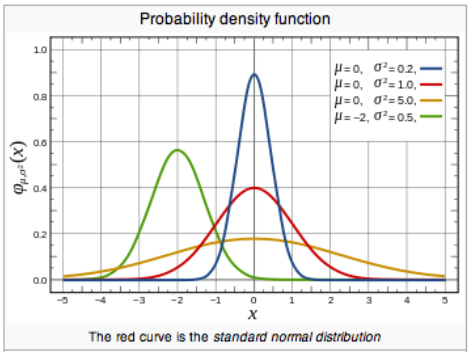
- Horloge 1 : il est 14h23mn à 2 mn près
- Horloge 2 : il est 14h19mn à 5 mn près

The BLUE is: $14H19\text{mn}32s$ at $+/-$ $10\sqrt{29}$ mn$(\approx 1,85$ mn$)$
the mean at +/- the variance

Basic recalls of Probability & Statistics
(Random variables, continuous & discrete)

The probability density function (pdf) in case of a Normal (or Gaussian) distribution:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(x-\bar{x})^2}{2\sigma^2})$$

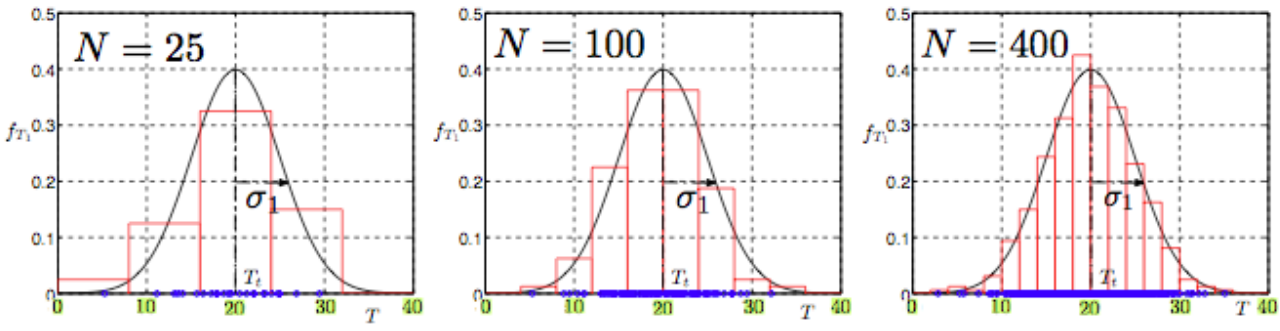


The mean (expected value):

$$\mu \equiv E(\hat{X}) = \int_a^b x f(x) dx \approx \frac{1}{N} \sum_{i=1}^{N} \hat{X}_n$$

The variance: $Var(\hat{X}) = E[(\hat{X} - \mu)^2]$

The standard deviation: $\sigma = (Var(\hat{X}))^2$



Plot: $\mu = 20$ mn $\sigma = 5^2$ mn

Calculation of the BLUE

1) The BLUE (called the estimate or the « analysis ») satisfies a linear relationship:

$$T_k = (1-k)T_1 + kT_2$$

2) Errors are supposed unbiased and uncorrelated, hence the variance writes:

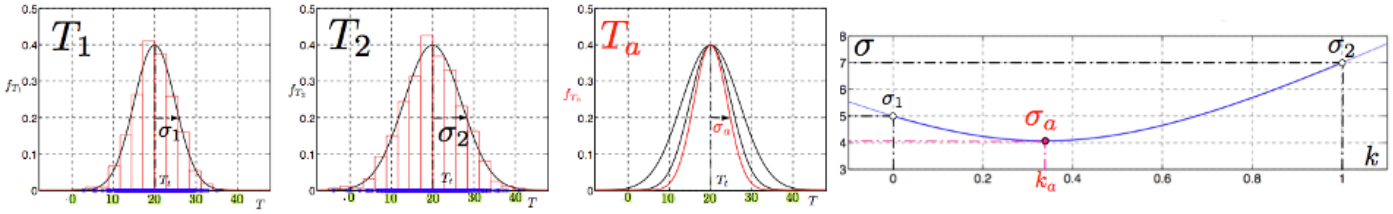
$$\sigma_a^2 = (1-k)^2\sigma_1^2 + k^2\sigma_2^2$$

3) The estimate minimizes the variance:

$$k_a = \frac{\sigma_2^{-2}}{\sigma_1^{-2} + \sigma_2^{-2}}$$

Finally, we obtain the BLUE / estimate / analysis:

$$\boxed{T_a = \frac{\alpha_1 T_1 + \alpha_2 T_2}{\alpha_1 + \alpha_2}} \quad \text{with} \quad \alpha_i = \frac{1}{\sigma_i^2}\ ,\ i = 1, 2. \quad \text{and} \quad \alpha_a = \frac{1}{\sigma_a^2} = \alpha_1 + \alpha_2$$



Equivalency between the BLUE and an optimal solution (the variational one)

*) The BLUE: $T_a = \dfrac{\alpha_1 T_1 + \alpha_2 T_2}{\alpha_1 + \alpha_2}$ minimizes the following cost function:

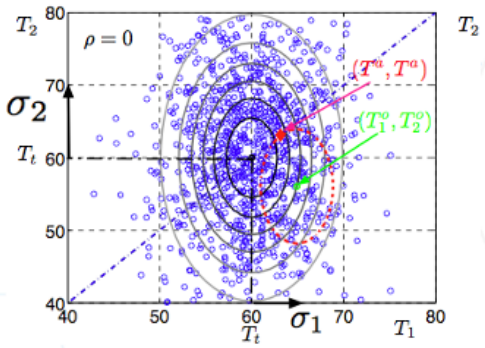
$$\boxed{j(T) = \frac{1}{2}\frac{(T-T_1)^2}{\sigma_1^2} + \frac{1}{2}\frac{(T-T_2)^2}{\sigma_2^2}}$$

*) One remark that the BLUE (obtained under the assumption of uncorrelated errors), minimizes the following cost function too:

$$j(T) = \frac{1}{2}(T-T_1, T-T_2)\begin{pmatrix} \sigma_1^2 & \rho_{12}\sigma_1\sigma_2 \\ \rho_{12}\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix}^{-1}\begin{pmatrix} T-T_1 \\ T-T_2 \end{pmatrix}$$

i.e. with a-priori correlated errors ! (see the correlation coefficients in the extra diagonal terms).

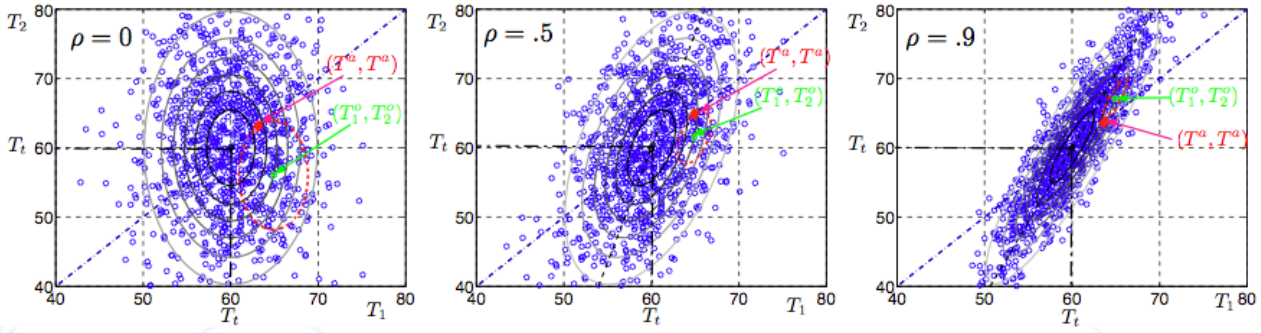
*) T1 and T2 are two measurements uncorrolated: the correlation coefficient vanishs.



$$\rho(T_1, T_2) = \frac{cov(T_1, T_2)}{\sigma_1 \sigma_2} = 0$$

*) T1 and T2 are two measurements corrolated:

$$\rho(T_1, T_2) = \frac{cov(T_1, T_2)}{\sigma_1 \sigma_2} \neq 0$$



How to improve the estimate ?
How to assimilate extra measurements ?

Assumption.

One knowns the « accuracy » of each measurement
i.e. each variance is known (or equivalently the standard deviation):

$$\sigma_i \ , \ i = 1, .., m \ (\text{for } m \text{ measurements}) \text{ given.}$$

Next, each extra measurement may « enhance the accuracy « of the estimate in the sense:

$$\frac{1}{\sigma_a^2} = \frac{1}{\sigma_1^2} + ... + \frac{1}{\sigma_m^2}$$

In fact, each extra measurement makes decrease the dispersion from the mean,
but not necessarily the accuracy of the estimate if the extra measurements are not accurate !...

The BLUEstimate writes:

$$T_a = \frac{1}{\sum_{i=1}^{m} \frac{1}{\sigma_i^2}} \left(\sum_{i=1}^{m} \frac{1}{\sigma_i^2} T_i \right) \text{ with } \frac{1}{Var(T_a)} = \sum_{i=1}^{m} \frac{1}{\sigma_i^2}$$

Finally, what time is it ?

Chapter 5

VDA for time-dependent PDE systems (parabolic, hyperbolic)

This chapter aims at extending the VDA formulation to unsteady PDE systems, parabolic or hyperbolic, non-linear or not. The calculations derived in the present chapter are formal only.
The VDA algorithm obtained for a time-dependent system is classically called *4D-var*. (Actually it is still called "4D-Var" even if the model is not 3D in space...).
In this chapter, the "4D-var algorithm" is derived. Next, some useful variants are derived too : 3D-var and incremental version.

The outline of this chapter is as follows[1].

Contents

[1]Recall that the sections indicated with a * are "to go further sections". They can be skipped in a first reading or if the reader is not particularly interested in deeper mathematical basis, mathematical proofs.

5.1 The direct model

Let us present a few classical examples of unsteady PDE models which could be "inverted" by the VDA method. Next, the considered general direct model is presented.

5.1.1 Classical PDE models

Linear parabolic: the diffusion equation

The typical and simplest linear parabolic model is the 'heat equation'. This equation models any diffusion phenomena in a homogeneous media. (Recall that the diffusion phenomena occurs in almost all applications : fluids, structures, chemical, images, finance etc).

Non-linear parabolic / hyperbolic equation: Burgers equations

The Burgers equation is a scalar non-linear advection equation (non-linear advection term); its viscous version considers a diffusive term.
The unknown is $u(x,t)$ the fluid velocity at point x and time t.
The forward (direct) model reads as follows.Find $u(x,t)$ which satisfies:

$$\left\{ \begin{array}{ll} \partial_t u(x,t) - \nu \partial^2_{xx} u(x,t) + u \partial_x u(x,t) = f(x,t) & \text{in }]0,L[\times]0,T[\\ u(x,0) \ = \ u_0(x) & \text{in }]0,L[\end{array} \right. \tag{5.1}$$

with boundary conditions.
For $\nu > 0$ the equation is parabolic; for $\nu = 0$ it is hyperbolic.

The Burgers equation ($\nu = 0$) is the most basic non-linear hyperbolic scalar equation, very instructive to understand hyperbolic system solution behaviors.

Non-linear hyperbolic system: The Saint-Venant equations (shallow-water model)*

This is a "to go further section"
A model widely used for geophysical shallow flows, or industrial film flows, is the shallow-water model (also called Saint-Venant equations in its 1D version). It offers a very interesting "reduced dimension" approach for fluid flows occurring in thin geometries i.e. geometries with a ratio between the characteristics depth and the characteristic length very small: $\varepsilon = H/L << .$ ε lower than 10^{-2} in practice.

Such geometries are common both in industrial flows (eg coating, thin films on a structure) and in geophysical flows (eg oceans, rivers, glaciers, lavas and historically for atmosphere). For environmental flow modeling, we refer the reader to [39].

Shallow-water models are non-linear hyperbolic systems, almost identical to the Euler's equations in compressible fluid mechanics.
A first derivation of the shallow equations has been done by Barre de Saint-Venant (1871) for river hydraulics. Researches are still active in the topic (in order to include non-linear turbulence terms etc). Roughly, these equations are obtained as follows. The Navier-Stokes equations free-surface, incompressible, are depth-integrated; next, one develop asymptotic expansions with respect to the small

(geometric) parameter ε. At the end, instead of a 3d Navier-Stokes free-surface model (mobile computational domain), we obtain a 2d model posed in a fixed domain; thus the reduction of dimension and the computational time saving.

The variable are classically (h, q) the depth and the discharge of the flow. With such an assumption the pressure is not a direct variable anymore (h plays somehow the role of p). The pressure value is nearly the hydrostatic value $\rho g h$.

Let us present the 2d shallow water equations in their conservative formulation. The state variables are the water depth h and the local discharge $\mathbf{q} = h\mathbf{u}$, where $\mathbf{u} = (u, u)^T$ is the depth-averaged velocity vector. On a 2d domain Ω and for a computational time interval $[0, T]$, the equations are:

$$\left\{ \begin{array}{lll} \partial_t h \ + \ \mathrm{div}(\,\mathbf{q}\,) \ = \ 0 & & \text{in } \Omega\times]\,0,T] \\ \partial_t \mathbf{q} \ + \ \mathrm{div}\left(\frac{1}{h}\,\mathbf{q}\otimes\mathbf{q}\right) \ + \ \frac{1}{2}g\nabla h^2 + \ gh\nabla z_b \ = \ S_f & & \text{in } \Omega\times]\,0,T] \\ h(0) \ = \ h_0, \qquad \mathbf{q}(0) \ = \ \mathbf{q}_0 & & \\ + \text{ boundary conditions} & & \end{array} \right. \tag{5.2}$$

where g is the magnitude of the gravity, z_b the bed elevation, h_0 and q_0 the initial conditions for the state variables. The quantity $c = \sqrt{gh}$ denotes the local wave celerity.

In river hydraulics, we introduce the *empirical* source term S_f modeling the *friction* at bottom. It reads:

$$S_f = g\frac{n^2\|\mathbf{q}\|_2}{h^{7/3}}$$

It is the Manning-Strickler law, with the scalar parameter n to be tuned... Classically it is tuned by hand but potentially using the present optimal control process ! (see numerical examples in next section).

Without the friction term S_f, these equations constitute a non-linear hyperbolic system (see your course on the subject). They are very similar to the Euler equation (compressible flow, unviscous, in a fixed domain).

This system must be closed with boundary conditions. Typical B.C. are as follows:
- at inflow, inflow discharge is imposed.
- at outflow, incoming characteristic are imposed (or rating curves, or at worth Neumann conditions),
- at obstacle ('lateral boundaries'), wall conditions are imposed: $u.n = 0$.

Let us point out that the solution $(h, \mathbf{q})$ of the shallow-water system can develop discontinuities and shocks in finite time and even with a regular initial condition. Hence, the solution can locally *be not* differentiable with respect to "parameters" (e.g. the boundary condition) ...
Keeping in mind this warning, we will however develop an adjoint approach on this system.

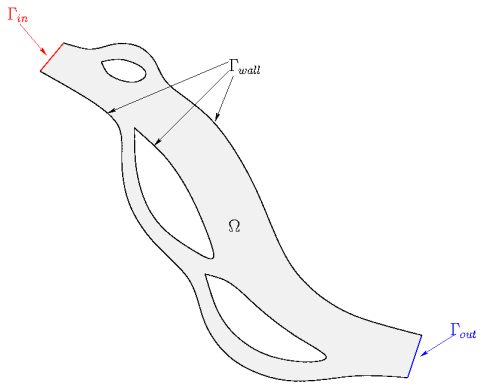


Figure 5.1: Example of application of the 2d shallow water model to river hydraulics (3d free-surface flow computed in a 2d fixed domain). The computational domain and its boundaries.

5.1.2 The general (and formal) direct model

Let V be a Hilbert space (e.g. the Sobolev space $H_0^1(\Omega)$). Let us assume that we have a first order in time PDE model e.g. the heat equation, the Navier-Stokes equations or the Saint-Venant equations. Then, we have to consider the following functional space:

$$W(0,T) = \{f \ s.t. \ f \in L^2(0,T;u), \ \partial_t f \in L^2(0,T;V')\}$$

As previously, the state of the system is denoted by y. We have: $y \in W(0,T)$.

Let $\mathcal{U}$ be the space of control parameters. it is supposed to be a Hilbert space. Since parameters can depend on time, we introduce the space : $\mathcal{U}_T = L^2(0,T;\mathcal{U})$. It is space of time dependent functions with values in $\mathcal{U}$; functions power 2 integrable. $\mathcal{U}_T$ is a Hilbert space with the norm defined from the scalar product:

$$\langle u_1, u_2 \rangle_{\mathcal{U}_T} \ = \ \int_0^T \langle u_1(t), u_2(t) \rangle_{\mathcal{U}} dt$$

The direct model reads:

$$\boxed{(\mathcal{D}) \left\{ \begin{array}{l} \text{Given } (y_0(x), u(x,t)) \in H \times \mathcal{U}_T, \text{ find } y(x,t) \in W(0,T) \text{ such that :} \\ \partial_t y(x,t) \ + \ A(u(x,t)\,; y(x,t)) \ = \ 0 \qquad \text{in } \Omega \times]0,T[\\ y(0) \ = \ y_0 \text{ in } \Omega \end{array} \right.} \tag{5.3}$$

where $A : \mathcal{U} \times V \longrightarrow V'$ is a partial derivatives operator, $y_0 \in H$ is the initial condition, and $u(t) \in \mathcal{U}$ is the parameter at time t.

Of course, the solution $y(x,t)$ depends on the I.C. $y_0(x)$ and the parameter $u(x,t)$.
Let us consider the following control variable:

$$\boxed{\mathbf{c}(x,t) = (y_0(x), u(x,t))^T} \tag{5.4}$$

The control $\mathbf{c}(x,t)$ belongs to the space: $\mathcal{C} = H \times \mathcal{U}_T$.
In the following, depending on the context, the state is denoted as follows: $y(\mathbf{c};t)$, $y(t)$, $y(\mathbf{c})$ or simply y.

We assume that the direct model is well posed in the following sense:

Assumption 5.1. *Given $\mathbf{c} \in \mathcal{C}$ and $T > 0$, it exists a unique function $y \in W(0,T)$ solution of the problem $(\mathcal{D})$. Furthermore, this unique solution depends continuously of $\mathbf{c}$.*

In other words,we assume the model operator defined by:

$$\boxed{\mathcal{M} : \mathcal{C} \to V \ : \mathbf{c}(x,t) = (y_0(x), u(x,t)) \mapsto y(\mathbf{c}(x,t); x, t)} \tag{5.5}$$

is continuous for all $t \in]0,T[$.

Furthermore, let us assume the state is differentiable with respect to the control parameters.

Assumption 5.2. *The operator model $\mathcal{M}(\mathbf{c})$ is continuously differentiable for all $t \in]0,T[$.*

Under the assumption above, formally one can write:

$$y(\mathbf{c} + \delta\mathbf{c}; t) \ = \ y(\mathbf{c}; t) + \frac{dy}{dk}(\mathbf{c}; t) \cdot \delta\mathbf{c} + o_0(\|\delta\mathbf{c}\|_{\mathcal{C}}) \tag{5.6}$$

This differentiability property is necessary to calculate the cost function gradient, but it is not always satisfied (in particular everywhere). Typically, non-linear hyperbolic systems (e.g. the shallow-water system) may generate non-differentiable, even non-continuous, solution with respect to some input parameters...
Let us recall that a powerful mathematical result to prove the differentiability is the implicit function theorem (roughly, the operator and the linearized model have to be C^1).

5.2 The cost function

5.2.1 Misfit to the time-dependent observations: averaging in time

A typical observation function J is defined as follows :

$$\boxed{J(\mathbf{c}\,; y(\mathbf{c})) \ = \ \int_0^T \Big\| \, C(y(\mathbf{c}\,; t)) - z^{obs}(t) \Big\|_O^2 \, dt \ + \ \alpha_{reg} J_{reg}(\mathbf{c})} \tag{5.7}$$

with J_{reg} a regularization term to be determined.
The choice of its expression depends on the modeling problem and/or the available priors on the solution (see below).

The definition of J is similar to those defined in the steady-state case excepted the integration (averaging) in time.

The cost function j to be minimized is defined from the observation function like in the steady-state case:

$$\boxed{\begin{array}{c} j(\mathbf{c}) = J(\mathbf{c}\,; y(\mathbf{c})) \\ \text{where } y(\mathbf{c}) \equiv y^{\mathbf{c}} \text{ is the (unique) solution of the direct model.} \end{array}} \tag{5.8}$$

We have: $j : \mathcal{C} \to \mathbb{R}$.

Pointwise observations In practice observations are often point-wise, moreover often very sparse. Indeed, it is common to have sensors at few locations only.

Moreover if these observations are provided as time-series, the misfit observation function reads:

$$\boxed{J_{misfit}(\mathbf{c}\,;y(\mathbf{c})) \;=\; \sum_{n=1}^{N} \Big\| \, C_n(y(\mathbf{c};t_n^{obs})) - z_n^{obs} \Big\|_O^2}$$

where C_n is the observation operator (linear or not),
z_n^{obs} is the measurement at instant t_n^{obs}, $n = 1, \ldots, N$. The complete dataset is: $z^{obs} = \{z_n^{obs}\}_{n=1,\ldots,N}$.

5.3 The optimization problem & regularization terms

As already defined in the previous chapter, a Variational Data Assimilation (VDA) formulation is nothing else than an optimal control problem with the cost function measuring the discrepancy between the model output and observations, see Fig. **??**.

5.3.1 The optimization problem

If considering the typical observation function defined by (5.7) and the corresponding cost function j, than the VDA problem reads like in the steady-state case:

$$\boxed{\left\{ \begin{array}{l} \text{Find } \mathbf{c}^* = (y_0, u)^{T*} \in H \times \mathcal{U}_T \text{ such that:} \\ j(\mathbf{c}^*) = \min\limits_{H \times \mathcal{U}_T} j(\mathbf{c}) \end{array} \right.} \tag{5.9}$$

Note that in the present context, the control parameter vector is a-priori time-dependent.
Therefore its discrete version may be an extremely large vector...
As a consequence, the gradient of the cost function must likely be computed by introducing the adjoint model.

5.3.2 On the regularization term

The regularization term J_{reg} act differently, depending on its definition.
For example it can:
- attract the minimization algorithm to a background value y_b, hopefully relatively good, by defining J_{reg} as follows:

$$J_{reg}(\mathbf{c}) = \Big\| \, y_0 - y_b \Big\|_N^2 \tag{5.10}$$

with $N = Id$ or eg. a low-band filter.
- simply regularize the control variable u by defining J_{reg} eg. as follows:

$$J_{reg}(\mathbf{c}) = \Big\| \, \nabla u \Big\|^2 \text{ or } J_{reg}(u) = \Big\| \, \Delta u \Big\|^2 \tag{5.11}$$

In theses cases, N denotes (semi-norm) of $H^1(\Omega)$ or the Laplace operator respectively.

Recall that such Tikhonov like regularization terms enhance the "convex feature" of the cost function; please refer to the discussion in the steady-state case.
The reader may consult e.g. the book [25] for more details on the regularization of optimization problems and the Tikhonov regularization in particular.

Exercice 5.3. *Suggest regularization terms for your practical problem.*

5.4 The linear tangent model and the resulting gradient

5.4.1 The linear tangent model

Let us recall that the (unique) state of the system $y(\mathbf{c})$ is assumed to be differentiable with respect to $\mathbf{c}$.

Given a perturbation $\delta\mathbf{c} \in \mathcal{C}$, the derivative of the state $y(\mathbf{c})$ in the direction $\delta\mathbf{c}$ (Gateaux's derivative) satisfies:

$$dy^{\delta\mathbf{c}} \equiv \frac{dy}{d\mathbf{c}}(\mathbf{c}) \cdot \delta\mathbf{c} \;=\; \frac{\partial y}{\partial y_0}(\mathbf{c}) \cdot \delta y_0 \;+\; \frac{\partial y}{\partial u}(\mathbf{c}) \cdot \delta u \tag{5.12}$$

Let us derive the direct model $(\mathcal{D})$ with respect to $\mathbf{c}$ in a direction $\delta\mathbf{c}$ (at the "point" $y(\mathbf{c})$). We obtain the so-called tangent linear model :

$$(\mathcal{LT}) \begin{cases} \text{Given } \mathbf{c} = (y_0, u) \in H \times \mathcal{U}_T \text{ and } y(\mathbf{c}) \in W(0,T) \text{ solution of the direct problem } (\mathcal{D}), \\ \text{given } \delta\mathbf{c} = (\delta y_0, \delta u) \in H \times \mathcal{U}_T, \text{ find } dy^{\delta\mathbf{c}} \in W(0,T) \text{ such that:} \\ \partial_t\, dy^{\delta\mathbf{c}}(t) \;+\; \frac{\partial A}{\partial y}(u(t); y(t)) \cdot dy^{\delta\mathbf{c}}(t) \;=\; -\,\frac{\partial A}{\partial u}(u(t); y(t)) \cdot \delta u(t) \qquad \forall t \in \,]0,T[\\ dy^{\delta\mathbf{c}}(0) \;=\; \delta y_0 \end{cases} \tag{5.13}$$

The linear tangent model describes the dynamic of the derivative of the state (solution of $(\mathcal{D})$) for a perturbation $\delta\mathbf{c}$ given.

Remark 5.4. *If the operator A is linear with respect to the state variable y, then we have:* $\dfrac{\partial A}{\partial y}(u(t), y(t)) \cdot dy(t) = A(u(t), dy(t))$ *and obviously the linearized tangent model is the same as the direct model (modulo the initial condition and the right hand side).*

In order to compute $dy^{\delta\mathbf{c}}$, we can solve the linearized problem $(\mathcal{LT})$. But for a different perturbation $\delta\mathbf{c}^*$, the linear tangent model $(\mathcal{LT})$ must be solved again to obtain $dy^{\delta\mathbf{c}^*}$!
Like in the steady-state case (previous chapter) that is the reason why the adjoint equations are introduced; they allow to circumvent this drawback.

5.4.2 The gradient expression depending on the linearized state

Remark 5.5. - *Recall that in order to solve numerically the minimization problem, we will use a descent algorithm. Hence, one need to evaluate the differentiable of j at any point $\mathbf{c}$ and in any direction $\delta\mathbf{c}$ i.e. $j'(\mathbf{c}) \cdot \delta\mathbf{c}$ for any $\delta\mathbf{c}$. It is what we call classically but unfairly "the gradient". Indeed, it is actually the differentiable of j. The terminology gradient usually refers to a vector in a finite*

dimension space.

- Since the control variables has two distinct components, $\mathbf{c} = (y_0, u)$*, then the "gradient"* $j'(\mathbf{c}) = \frac{dj}{d\mathbf{c}}(\mathbf{c})$ *reads as follows (recall it is an element of* $\mathcal{L}(H \times \mathcal{U}_T; \mathbb{R})$*) :*

$$\boxed{"j'(\mathbf{c}) = \frac{dj}{d\mathbf{c}}(\mathbf{c}) = \nabla j(\mathbf{c}) = (\frac{\partial j}{\partial y_0}(\mathbf{c}), \frac{\partial j}{\partial u}(\mathbf{c}))^T"}$$

while:

$$\boxed{j'(\mathbf{c}) \cdot \delta\mathbf{c} = \frac{\partial j}{\partial y_0}(\mathbf{c}) \cdot \delta y_0 + \frac{\partial j}{\partial u}(\mathbf{c}) \cdot \delta u}$$

A straightforward differentiation of the cost function (3.7) gives :

$$j'(\mathbf{c}) \cdot \delta\mathbf{c} \;=\; \partial_{\mathbf{c}} J(\mathbf{c}\,; y(\mathbf{c})).\delta\mathbf{c} + \partial_y J(\mathbf{c}\,; y(\mathbf{c})).dy^{\delta\mathbf{c}}$$

Let us consider the functional J defined by (5.7) with a linear observation operator C and the regularization term J_{reg} defined by (5.10). In this case, on has:

$$j'(\mathbf{c}) \cdot \delta\mathbf{c} \;=\; \int_0^T \Big\langle\, C^*\Lambda_{\mathcal{O}}(C\,y(t) - z^{obs}(t)),\, dy^{\delta\mathbf{c}}(t) \,\Big\rangle_{V'\times V} dt \;+\; \big\langle N(y_0 - y_b), \delta y_0 \big\rangle_H \tag{5.14}$$

where $\langle \cdot, \cdot \rangle_{V'\times V}$ denotes the duality product.

The operator $\Lambda_{\mathcal{O}}$ is the canonical isomorphism from $\mathcal{O}$ into $\mathcal{O}'$,
$C^* \in \mathcal{L}(\mathcal{O}', V')$ is the adjoint operator of C. The latter is simply defined by: $\forall\, \eta \in \mathcal{O}', \quad \forall\, \xi \in V \qquad \big\langle\, C^*\eta,\, \xi \,\big\rangle_{V'\times V} \;=\; \big\langle\, \eta,\, C\xi \,\big\rangle_{\mathcal{O}'\times\mathcal{O}}$.
In discrete dimension it would be the Identity.

Like in the steady-state case (see the previous chapter), the resulting expression of $j'(\mathbf{c}) \cdot \delta\mathbf{c}$ depends explicitly on $dydy^{\delta\mathbf{c}}$.
In a descent algorithm, one need to evaluate $j'(\mathbf{c}) \cdot \delta\mathbf{c}$ for a large number of directions $\delta\mathbf{c}$. Therefore the present expression requires to solve the linear tangent model ($\mathcal{LT}$) as much as the (discrete) dimension of $\delta\mathbf{c}$. Recall that in practice, this approach is affordable only for a very small number of discrete control variables. To circumvent this problem, the adjoint equations are introduced.

5.5 The adjoint model, gradient and optimality system

Let us derive the adjoint equations of the problem. Recall that the adjoint equations enable to obtain an expression of $j'(\mathbf{c}) \cdot \delta\mathbf{c}$ linearly dependent of $\delta\mathbf{c}$, therefore an explicit gradient expression for *any* direction $\delta\mathbf{c}$.

5.5.1 The adjoint equations

In the unsteady case, the starting point to derive the adjoint equations is to write:

$$\int_0^T < (\mathcal{L}\mathcal{T}), p >_{V'\times V} dt$$

next, applying an integration by part in time.
This gives:

$$\int_0^T \left\langle \partial_t\, dy(t), p(t)\right\rangle_{V'\times V} dt \;+\; \int_0^T \left\langle \tfrac{\partial A}{\partial y}\cdot dy(t), p(t)\right\rangle_{V'\times V} dt + \int_0^T \left\langle \tfrac{\partial A}{\partial u}\cdot \delta u, p(t)\right\rangle_{V'\times V} dt \;=\; 0$$

After the integration by part in time, and by making appear the adjoint operators, it comes :

$$\begin{aligned}\int_0^T \left\langle \partial_t\, p(t) - [\tfrac{\partial A}{\partial y}]^* p(t), dy(t)\right\rangle_{V'\times V} dt \;=\; \\ \left\langle p(T), dy(T)\right\rangle_V \;-\; \left\langle p(0), \delta y_0\right\rangle_V \;-\; \int_0^T \left\langle [\tfrac{\partial A}{\partial u}]^* p(t), \delta u(t)\right\rangle_{\mathcal{U}'\times\mathcal{U}} dt\end{aligned} \tag{5.15}$$

Next the "adequate equations" to make vanish all the terms in $dy(t)$, are introduced; this is the adjoint equations.
They read :

$$(\mathcal{A})\;\begin{cases} \text{Given } \mathbf{c} = (y_0, u) \in \mathcal{C} \text{ and } y^k \in W(0,T) \text{ be the unique solution of the direct problem } (\mathcal{D}),\\ \text{ find } p \in W(0,T) \text{ such that:}\\ \partial_t\, p(t) - \left[\frac{\partial A}{\partial y}(u(t); y^k(t))\right]^* p(t) = \partial_y J(\mathbf{c}\,; y^k(t)) \quad \forall t \in\,]0,T[\\ p(T) \;=\; 0 \end{cases} \tag{5.16}$$

In the present cost function definition case, we have:

$$\partial_y J(\mathbf{c}\,; y^k(t)) = \int_0^T C^* \Lambda_{\mathcal{O}} \Big(C\, y^k(t) - z^{obs}(t)\Big)\; dt \tag{5.17}$$

Let us recall that if the weak form of the direct model reads: $a(u; y, z) = l(u; z)\ \forall z$; then the corresponding adjoint equation is: $\partial_y a(u; y, p)\cdot z = \partial_y J(u; y)\cdot z\ \forall z$.
Therefore the weak form of the adjoint operator $\left[\frac{\partial A}{\partial y}(u(t); y^k(t))\right]^* p(t)$ reads:

$$< \left[\frac{\partial A}{\partial y}(u(t); y^k(t))\right]^* p(t), z(t) >= \partial_y a\left(u(t); y^k, p(t)\right)\cdot z(t) \tag{5.18}$$

Finally, we obtain the following adjoint model expression:

$$\begin{cases} \text{Given } \mathbf{c} = (y_0, u) \in \mathcal{C} \text{ and } y^k \in W(0,T) \text{ the unique solution of the direct problem } (\mathcal{D}),\\ \text{ find } p \in W(0,T) \text{ such that:}\\ \partial_t\, p(t) - \left[\frac{\partial A}{\partial y}(u(t); y^k(t))\right]^* p(t) = \partial_y J(\mathbf{c}\,; y^k(t)) \quad \forall t \in\,]0,T[\\ p(T) \;=\; 0 \end{cases} \tag{5.19}$$

Some important remarks. Let us point out that:

- the adjoint model is retroacting in time; its initial condition must be given at final time T.

- like in the steady-state case, by construction the adjoint model is linear (whatever if the direct model is linear or not).

- the RHS of the adjoint model is somehow a *nudging term* :

$$\boxed{C^*\Lambda_{\mathcal{O}}(Cy(t) - z^{obs}(t))}$$

It measures the discrepancy between the computed state and the observations (in the observations space).

5.5.2 The gradient expression

By combining equations (5.14), (5.15) (5.19), we obtain:

$$j'(\mathbf{c}) \cdot \delta\mathbf{c} \;=\; \big\langle N(y_0 - y_b) - p(0), \delta y_0 \big\rangle_H - \int_0^T \big\langle [\tfrac{\partial A}{\partial u}]^* p(t), \delta u(t) \big\rangle_{\mathcal{U}' \times \mathcal{U}} \, dt \tag{5.20}$$

The expression (5.20) of $j'(\mathbf{c}) \cdot \delta\mathbf{c}$ does not depend anymore on $dy^{\delta\mathbf{c}}$. By construction, it depends linearly on $\delta\mathbf{c}$. Since:

$$\boxed{j'(\mathbf{c}) \cdot \delta\mathbf{c} = \frac{\partial j}{\partial y_0}(\mathbf{c}) \cdot \delta y_0 + \frac{\partial j}{\partial u}(\mathbf{c}) \cdot \delta u \;,} \tag{5.21}$$

then the two components of the gradient (differential expression with respect to y_0 and $\mathbf{u}$) read :

$$\boxed{\left\{ \begin{array}{rcl} \partial_{y_0} j(\mathbf{c}) & = & -\, p^{\mathbf{c}}(0) \;+\; N(y_0 - y_b) \\ \partial_u j(\mathbf{c}) & = & -[\dfrac{\partial A}{\partial u}(u(t); y^{\mathbf{c}}(t))]^* \; p^{\mathbf{c}}(t) \end{array} \right.} \tag{5.22}$$

Observe that the gradient with respect to the initial condition equals the adjoint state at initial time (modulo the background term).

Note that if a regularization term on u, say $J_{reg}(u)$, is introduced in the definition fo J, then its derivatives appears in the expression of $\partial_u j(\mathbf{c})$, like in the steady-case. Please refer to the steady-state case chapter for details.

Exercice 5.6. *Consider the time-dependent case of your practical. Then:*
a) Write the adjoint equations; both in the weak and the classical form.
b) Write the gradient expression.

The 1st order optimality system

As previously (steady-state system in the previous chapter), we can define the optimality system. It is the set of equations characterizing the optimal solution: the state equation, the adjoint state equation and the first order necessary optimality condition (the gradient must vanishes).

5.6 The 4D-Var algorithm

5.6.1 The algorithm

The so-called 4D-var algorithm denotes the optimal control algorithm for unsteady PDEs systems (a-priori in 3 space-dimensions plus time therefore the 4D terminology). The adjoint model is time dependent, reverse in time.
Since the minimisation is performed by using an iterative descent algorithm (eg the quasi-Newton method BFGS), the algorithm reads as indicated in Fig. (5.2).
Given a *first guess* u^0, compute $\mathbf{c}^m$ making diminish the cost function j. To do so, at each iteration:

1) compute the cost function $j(\mathbf{c})$ by solving the direct model from 0 to T,

2) compute the gradient $j'(\mathbf{c})$ by solving the adjoint model from T to 0,

3) given the current iteration $\mathbf{c}^n$, the cost function value $j(\mathbf{c}^n)$ and the gradient value $\nabla j(\mathbf{c}^n)$, compute a new iteration$\mathbf{c}^{n+1}$ such that:

$$j(\mathbf{c}^{n+1}) < j(\mathbf{c}^n)$$

*) Iterations until convergence

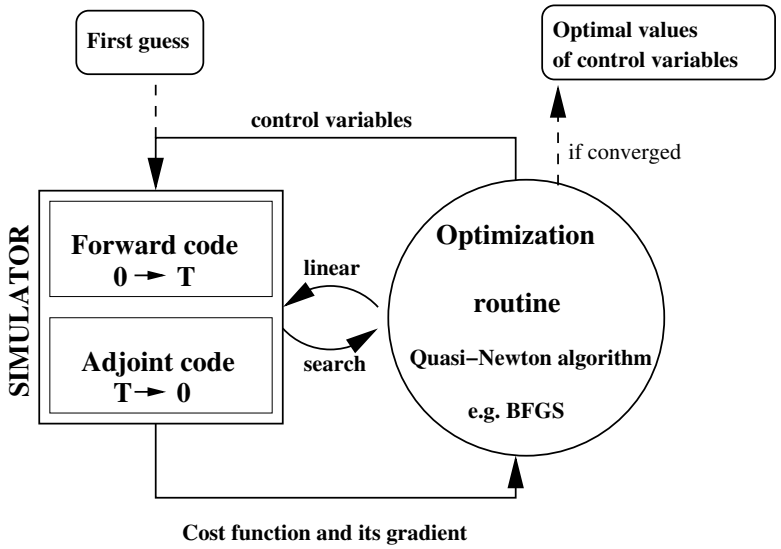


Figure 5.2: Optimal control algorithm for a time-dependent PDE model : the 4D-var algorithm.

5.6.2 A 4D-var algorithm: what for ?

The 4D-var algorithm can be used for different goals.

A) To estimate the value of uncertain or unknown input parameters (time dependent or not), it is an *identification problem.*

B) To *calibrate* the model in order to perform better predictions; forecasting is the goal.
In this case, the data assimilation proceeds by *"analysis cycles"*. Figure 5.3 represents one cycle.

1) The first stage is called the *"analysis step"*.
Observations of present and past time are assimilated to obtain the analysis i.e. the optimal state value (that is the optimal model trajectory).

2) Next stage consists to run the model in time: it is the *"forecasting step"*.
It is expected that if the model fits better the observations in the past, it will be more accurate for future.

Next, the forecast is used in the next analysis cycle etc

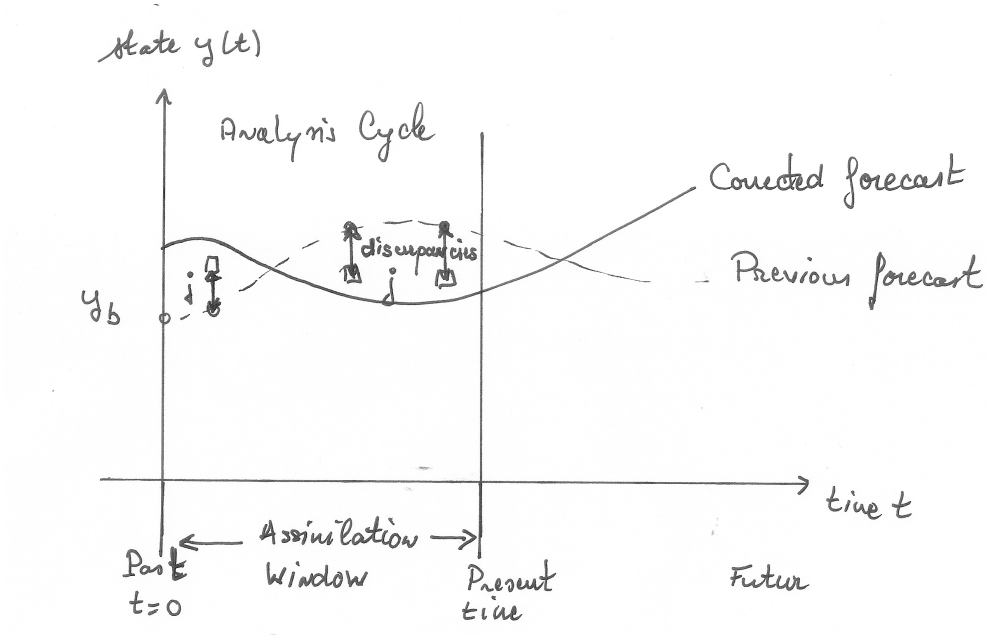


Figure 5.3: The 4D-var algorithm here used to identify the I.C. y_0. All observation within the assimilation time window are assimilated in the global least-square formulation (analysis step). Next, the resulting calibrated model is runned for prediction (forecasting step).

5.6.3 The local sensitivity analysis: a rich information

As mentioned above, the adjoint method enables the evaluation of $j'(\mathbf{c}) \cdot \delta\mathbf{c}$ independently of the linearized state $dy^{\delta\mathbf{c}}$.
This is an explicit expression of all the "gradient" components of $j'(\mathbf{c}) \cdot \delta\mathbf{c}$.
Computing a gradient, even without performing a descent algorithm, presents interest in a modeling point of view. Indeed, *the gradient $j'(\mathbf{c}) \cdot \delta\mathbf{c}$ represents the local sensitivity of the model output j with respect the parameters $\mathbf{c}$.*

The gradient helps to understand the parameter influences of the model and to improve the understanding of the system.

Nevertheless this gradient - sensitivity is local in the sense that it is regarding to the given point u^0 only (and regarding to the observations used too ....)

Let us remark that this idea of computing sensitivities in order to better understand both the model and the physics, *can be applied in a context without observations.*
Typically, it can be applied to cost functions depending on the state of the system only in view to study the system stability.
For example, it can be interesting to quantify the sensitivity of the following model output :

$$j(k) = \frac{1}{2} \int_{\omega} \|\nabla y\|^2_{\mathbb{R}^N} dx$$

with ω a particular subset of Ω.

5.6.4 A concluding remark

Data assimilation aims at fusing in an "optimal way" all available information: the "physics" of the phenomena (e.g. conservation laws), the parameters (generally empirical), the initial condition (particularly in geophysics), the in-situ measurements, the remote-sensed measurements (e.g. information extracted from satellite images acquired by various sensors), prior probabilistic behavior (covariance operators, see next section).

Additional measurement may be used to improve the "analysis", especially if its confidence (accuracy) can be estimated by expertise or a statistical method.

Estimating uncertain parameters of physical-based models may be addressed by "blind" statistical learning methods; the present physically-informed approach enables to it too.
Moreover it enables to assess the result in a physical point of view since the model constraint.
Moreover, the approach enable to combine the different mathematical "tools" including statistical learning e.g. through the prior values: first guess value or covariance operators...

5.7 The fundamental equations at a glance

The non-linear unsteady PDE model

$$(\mathcal{D})\left\{\begin{array}{l}\text{Given } (y_0(t),u(x,t))\in H\times\mathcal{U}_T,\ \text{find } y(x,t)\in W(0,T) \text{ such that :}\\ \partial_t\, y(x,t)\ +\ A(u(x,t)\,;y(x,t))\ =\ L(u(x,t)) \qquad \text{in } \Omega\times]0,T[\\ y(0)\ =\ y_0 \text{ in } \Omega\end{array}\right. \tag{5.23}$$

It is supposed to be well-posed.

The model operator $\mathcal{M}(\mathbf{c})$ is defined as: $\mathcal{M}(\mathbf{c})=y^{\mathbf{c}}(x,t)$.
The "operator model" $\mathcal{M}(\cdot)$ is a-priori non linear.

The control parameter is: $\mathbf{c}(x,t)=(y_0(x),u(x,t))$.

A **typical observation function** reads:

$$J(\mathbf{c}\,;y(\mathbf{c}))\ =\ \frac{1}{2}\int_0^T\Big\|\,C(y(\mathbf{c}\,;t))-z^{obs}(t)\Big\|_O^2\,dt\ +\ \alpha_0\frac{1}{2}\big\langle N(y_0-y_b),y_0-y_b\big\rangle_H+\ \alpha_u J^u_{reg}(u) \tag{5.24}$$

In a discrete version, the misfit to observations term reads:

$$J_{obs}(\mathbf{c}\,;y(\mathbf{c}))\ =\ \frac{1}{2}\sum_{n=1}^{N}\Big\|\,C_n(y(\mathbf{c};t_n^{obs}))-z_n^{obs}\Big\|_O^2$$

The **cost function** is defined as:

$$\begin{array}{l}j(\mathbf{c})=J(\mathbf{c}\,;y(\mathbf{c}))\\ \text{where } y(\mathbf{c})\equiv y^{\mathbf{c}} \text{ is the (unique) solution of the direct model.}\end{array} \tag{5.25}$$

The **optimization problem** is:

$$\left\{\begin{array}{l}\text{Find } \mathbf{c}^*(x,t)=(y_0(x),u(x,t))^*\in H\times\mathcal{U}_T \text{ such that:}\\ j(\mathbf{c}^*)=\min\limits_{\mathbf{c}\in H\times\mathcal{U}_T} j(\mathbf{c})\end{array}\right. \tag{5.26}$$

The linear tangent model reads:

$$(\mathcal{LT})\left\{\begin{array}{l}\text{Given } \mathbf{c}=(y_0,u) \text{ and } y(\mathbf{c})\in W(0,T) \text{ the unique solution of the direct problem } (\mathcal{D}),\\ \text{given } \delta\mathbf{c}=(\delta y_0,\delta u),\ \text{find } dy\in W(0,T) \text{ such that:}\\ \partial_t\, dy(x,t)\ +\ \partial_y A(u(x,t);y(x,t))\cdot dy(x,t)\ =\ -\,\partial_u A(u(x,t);y(x,t))\cdot\delta u(t)+L'(u)\cdot\delta u(t) \qquad \forall t\in\,]0,T\\ dy(x,0)\ =\ \delta y_0(x)\end{array}\right. \tag{5.27}$$

The adjoint model reads:

$$(\mathcal{A})\left\{\begin{array}{l}\text{Given } \mathbf{c}=(y_0,u) \text{ and } y^k\in W(0,T) \text{ the unique solution of the direct problem } (\mathcal{D}),\\ \text{find } p\in W(0,T) \text{ such that:}\\ \partial_t\, p(x,t)-\Big[\partial_y A(u(x,t);y^k(x,t))\Big]^*p(x,t)=\partial_y J(\mathbf{c}\,;y^k(x,t)) \quad \text{in } \Omega\times]0,T[\\ p(x,T)\ =\ 0\end{array}\right. \tag{5.28}$$

The **gradient** components, i.e. the partial derivatives with respect to $y_0(x)$ and $u(x,t)$, read :

$$\left\{ \begin{array}{rcl} \partial_{y_0} j(\mathbf{c}(x,t)) & = & -\ p^{\mathbf{c}}(x,0) \ +\ \alpha_0 N(y_0 - y_b)(x) \\ \partial_u j(\mathbf{c}(x,t)) & = & [-\partial_u A(u; y^{\mathbf{c}}) + L'(u)]\ p^{\mathbf{c}}(x,t) + \ \alpha_u (J^u_{reg})'(u)(x,t) \end{array} \right. \tag{5.29}$$

5.8 Examples in geosciences *

We present below few numerical results extracted from research studies. The first two subsections present applications in river hydraulics (developed originally at INP Grenoble, then at INSA & Mathematics Institute of Toulouse by the author and his collaborators M. Honnorat, J. Marin etc , see their work in [24, 32]). Models are based on the 2d shallow-water equations. Such "toy test cases" demonstrate the capabilities of the VDA method applied to river hydraulics. The reference computational code in this topic is DassFlow (Data Assimilation for Free Surface Flows, [17]). We refer to the next chapter to learn how to validate the adjoint code and the full optimization process.
Next, the historical (and extremely complex) example is briefly presented: weather forecasting. For this illustration, the information and images have been extracted from the literature (see the cited sources and references).

5.8.1 Identification of the topography in a 2d shallow-water model

We perform what we call *twin experiments*. The principle of twin experiments is as follows. First, we generate data using the direct code only. Next, we add some white noise to the 'perfect' data generated by the model. Next, we start the optimal control process from a first guess and we try to recover the set of parameters which gave the data. After the validation of the computational software (see next chapter to known how to validate an adjoint code), such numerical experiments is the necessary last step to validate both the approach and the computational code.

The first twin experiment presented concerns the identification of the topography in a small scale and academic case. The domain is 30 m long and 4 m large, and the topography is defined by:

$$\begin{aligned} z_b(x,y) &= 0.9\exp\left(-\tfrac{1}{4}(x-10)^2\right)\exp\left(-(y-1)^2\right) \\ &+ 0.7\exp\left(-\tfrac{1}{8}(x-20)^2\right)\exp\left(-2(y-3)^2\right) \end{aligned} \tag{5.30}$$

The inflow boundary is at $x = 0$, the outflow boundary at $x = 30$. Boundaries $y = 0$ and $y = 4$ are walls. We use a rectangular structured mesh of dimension 90×20.
Bed roughness, defined by its Manning coefficient, is uniform ($n = 0.025$). (see the definition of the source term S_f in the shallow-water model). We impose a constant discharge $q^{in} = 8\ m^3/s$ at $x = 0$ and a constant water height $h_{out} = 1.4\ m$ at $x = 30$. We obtain a steady state solution after about 80 s of simulation.

Figure 5.4 shows the water height of this steady state solution and the topography.

From this steady state solution, we extract the forthcoming observations: h^{obs} and u^{obs} every $0.02\,s$ during $20\,s$ on each cell.
It means that in this first academic test case, we observe fully the (steady-state) flow !

The objective of this test case is to retrieve the topography. The first guess used is a flat bottom.

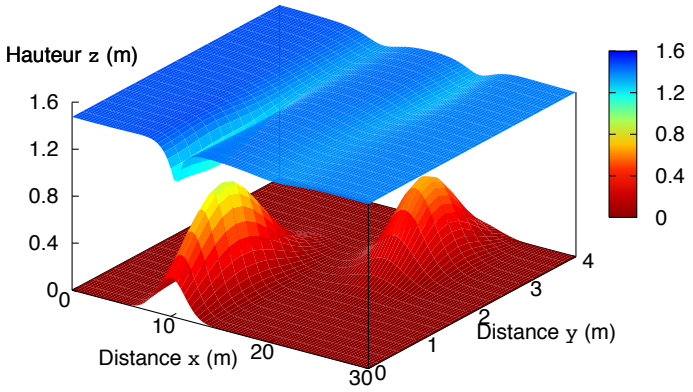
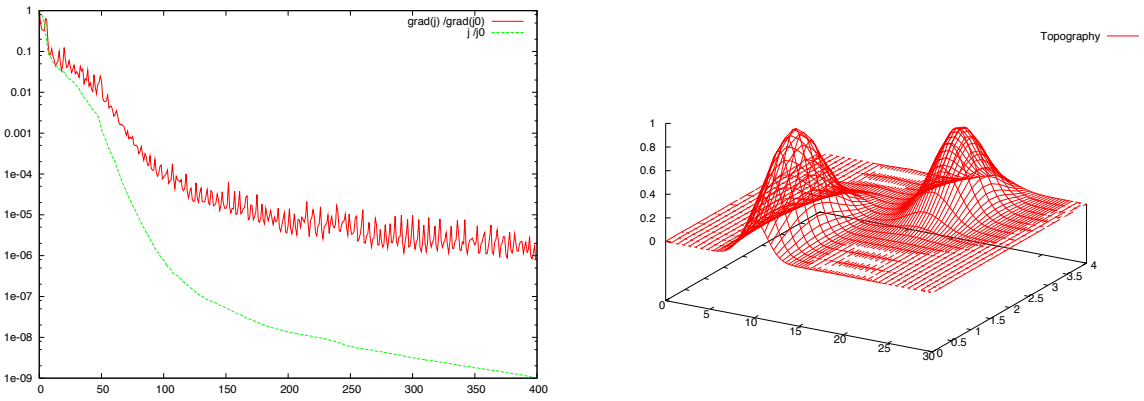


Figure 5.4: Identification of the topography. Topography and steady state elevation. From [24, 33].

We run the data assimilation process with the following cost function:

$$j_1(z_b) \;=\; \frac{1}{2}\int_0^T \left(\left\| \, h(t) - h^{obs}(t) \right\|_\Omega^2 + \left\| \, \mathbf{q}(t) - \mathbf{q}^{obs}(t) \right\|_\Omega^2 \right) dt \;, \tag{5.31}$$

Figure C.9 shows the cost function and the norm of its gradient normalized by its initial values, vs iterates (a) and the identified topography (b). We can notice that convergence is obtained and the reference topography is well retrieved.



(a) (b)

Figure 5.5: Value of the cost function and of the norm of its gradient, normalized by their initial values (a) and the identified topography (b)

This fully observed test case shows illustrate the variational data assimilation method in case of the identification of an underlying topography.

5.8.2 Identification of inflow boundary conditions in a 2d shallow-water model (flooding)

We consider a toy test case which includes many features of real river flows. The computational domain contains a main channel (river) and floodplains, see figures 5.6 and 5.7).

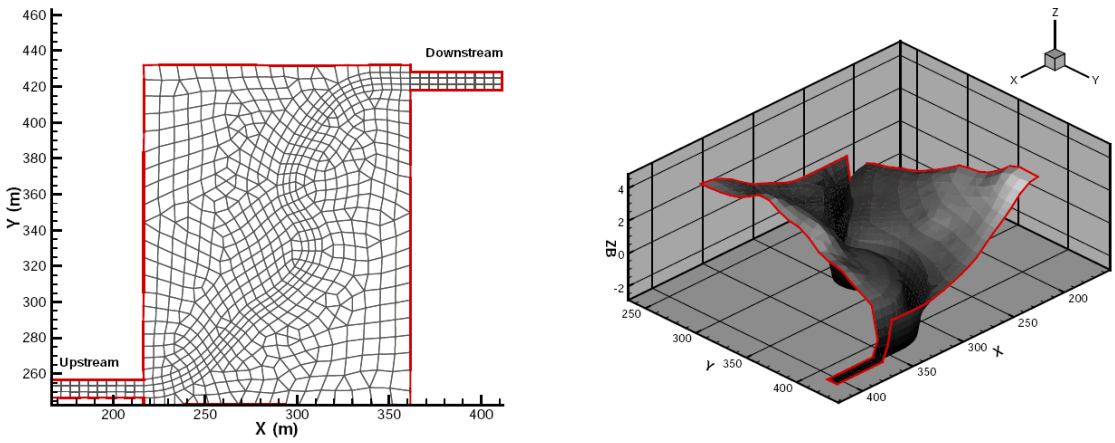
Again, the present test case is a twin experiment. At the inflow boundary, we set the inflow discharge shown if figure 5.8 (a) simulating a flood event.

Then we perform a forward run to generate observations at points 1 and 2 shown with black stars in figure 5.7(a).
Then, we suppose that the inflow discharge is constant ($4.95\, m^3 s^{-1}$), and we try to retrieve its real value by assimilating observations.

We present in Figure 5.8 the identified inflow discharge for different experiments. In Fig. 5.8(a), observations are h and $\mathbf{q}$ at each cell and each time step. In Fig. 5.8(b), observations are h at point 1 and (h, $\mathbf{q}$) at point 2, both at each time step. In Fig. 5.8(c), observations are h at point 1 only, but at each time step.

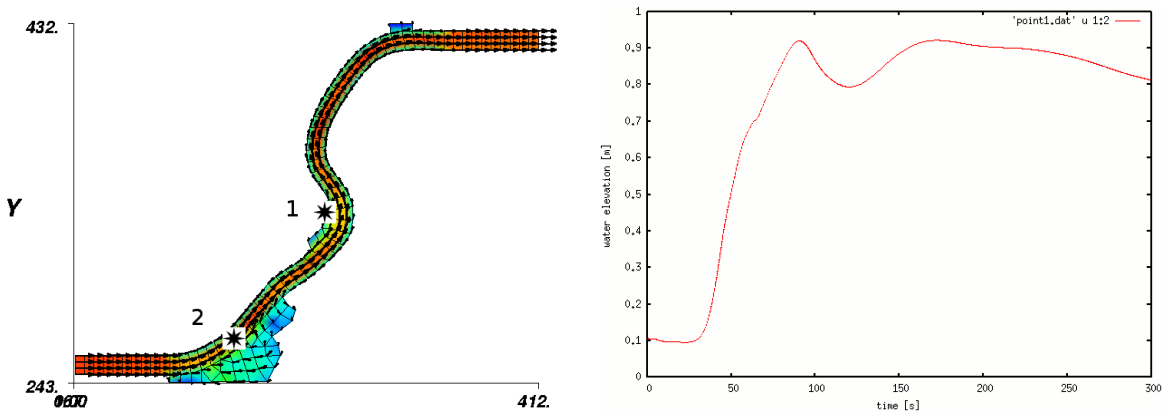
We can notice that the identified inflow discharge is good even with the observation of h at point 1 only.
In a practical point of view, such a test case show the ability of the method to identify inflow discharge in a river ...

One notice that the end of the flood event is not well identified. This is the "blind period" phenomena: for example in case (c), the inflow discharge after 270 s can not be identified because the information from the inflow boundary did not reach yet the gauge station.



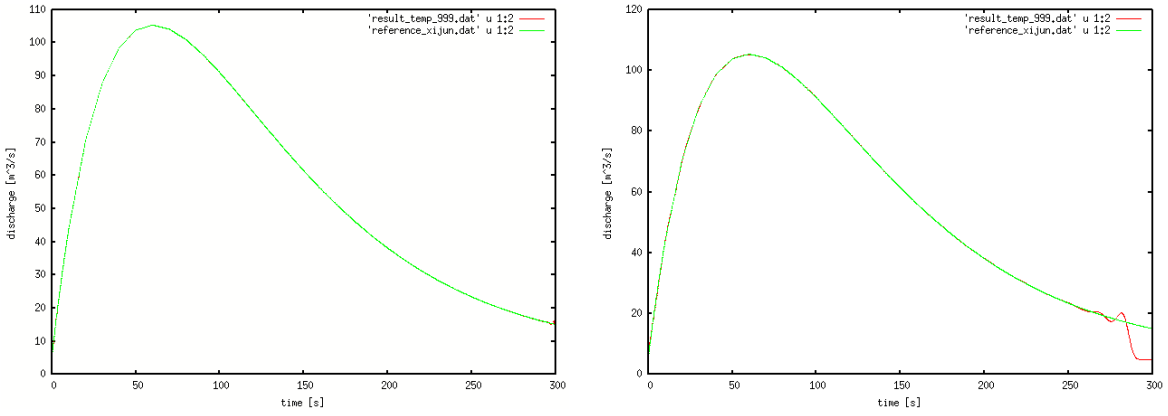
(a) (b)

Figure 5.6: Toy test case mesh (a) and bathymetry (b)

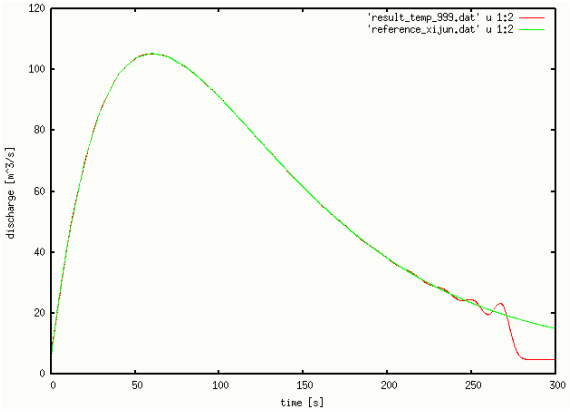


(a) (b)

Figure 5.7: Toy test case domain with mesurement points (a) and observation data available at point 1 (b)



(a) Observations everywhere (b) 2 observation points



(c) 1 observation point

Figure 5.8: In green : reference inflow discharge. In red : identified inflow discharge. (a): Observation of $(h, \mathbf{q})$ everywhere; (b): Observation of h at point 1 and $(h, \mathbf{q})$ at point 2. (c): Observation of h at point 1 only. From [24, 33]

5.9 Exercises: optimality systems of classical PDE models *

5.9.1 Viscous Burgers' equation

The viscous Burgers' equation is the 1d simplification of the Navier-Stokes momentum equation. It is a scalar non-linear advection diffusion equation (non-linear advection term). The unknown is $u(x,t)$ the fluid velocity at point x and time t.
The control variables we consider in the present example are: the initial condition u_0 and the velocity value at one boundary extremity; the latter is denoted by v.
The forward (direct) model reads as follows.
Given $\mathbf{c} = (u_0, v)$, find u which satisfies:

$$\left\{\begin{array}{ll} \partial_t u(x,t) - \nu \partial^2_{xx} u(x,t) + u \partial_x u(x,t) = f(x,t) & \text{in }]0,L[\times]0,T[\\ u(x,0) \ = \ u_0(x) & \text{in }]0,L[\\ u(0,t) = v(t) \ ; \ u(L,t) = 0 & \text{in }]0,T[\end{array}\right. \tag{5.32}$$

We assume we have m observations points of the flow, continuous in time. Then, we seek to minimize the following cost function:

$$j(\mathbf{c}) = \frac{1}{2} \int_0^T \sum_{i=1}^{m} |u(x_i) - u_i^{obs}|^2 dt$$

Exercice 5.7. *Write the optimality system corresponding to this data assimilation problem.*

5.9.2 Diffusion equation with non constant coefficients

We consider the diffusion equation (or heat equation) in an inhomogeneous media. Let u be the quantity diffused and $\lambda(x)$ be the diffusivity coefficient, non constant. The forward model we consider is as follows. Given λ and the flux φ, find u which satisfies:

$$\left\{\begin{array}{ll} \partial_t u(x,t) - \partial_x(\lambda(x) \partial_x u(x,t)) = f(x,t) & \text{in } \Omega\times]0,T[\\ u(x,0) \ = \ u_0(x) & \text{in } \Omega \\ -(\lambda(x) \partial_n u(x,t) = \varphi & \text{in } \Gamma_1\times]0,T[\\ u(x,t) = 0 & \text{in } \Gamma_0\times]0,T[\end{array}\right. \tag{5.33}$$

with $\partial\Omega = \Gamma_0 \cup \Gamma_1$.
We assume we have measurements of the quantity u at boundary Γ_1, continuously in time. Then, we seek to minimize the following cost function:

$$j(\mathbf{c}) = \frac{1}{2} \int_0^T \int_{\partial\Omega} |u(x) - u^{obs}|^2 ds dt$$

Exercice 5.8. *Write the optimality system corresponding to this data assimilation problem.*

Chapter 6

Dictionary-based model learning

The model learning approach which is presented in this section has been first proposed in [12][36]. This recent idea is here detailed for ODE systems. Next, it is briefly shown how to extend it to simple scalar PDE models e.g. the viscous Burgers equation.

This chapter has been written with the great help of Syver Agdestein, INSA student, during his summer 2019 internship

The outline of this chapter is as follows[1].

[1]Recall that the sections indicated with a * are to go further sections . They can be skipped in a first reading or if the reader is not particularly interested in deeper mathematical basis, mathematical proofs.

Contents

6.1 Learning ODE terms from a dictionary

6.1.1 Basic principle

Let us consider the general ODE system of unknown $\mathbf{x}(t)$, $\mathbf{x}(t) = [x_1(t), x_2(t), \cdots, x_N(t)]^T \in \mathbb{R}^N$:

$$\frac{d\mathbf{x}}{dt}(t) = \mathbf{f}\big(\mathbf{x}(t)\big),\ t \in [0, T] \tag{6.1}$$

with the initial condition:

$$\mathbf{x}(0) = \mathbf{x}_0,\ \mathbf{x}_0 \in \mathbb{R}^N \tag{6.2}$$

We have: $\mathbf{f} : \mathbb{R}^N \to \mathbb{R}^N$.

Let us assume that one has measurements of $\mathbf{x}(t)$ *and its derivatives* $\dot{\mathbf{x}}(t)$ *at time instants* $t_{1,} \cdots, t_M$.
The present model learning problem aims at *identifying* the unknown EDO expression, that is the *unknown expression of* $\mathbf{f}\big(\mathbf{x}(t)\big)$.

Note that for higher-order ODE, measuring $\mathbf{x}\dot{(}t)$ means measuring higher-order derivatives of $\mathbf{x}(t)$...

Example #1: the spring -mass system dynamic

Let a mass m be attached to a spring of constant k and friction coefficient ν. We denote by $x(\cdot)$ the mass displacement with respect to its equilibrium location. The second Newton law of motion provides the mass dynamics equation.

For small displacement: linear model. For small displacement, the linearized version is a sufficiently accurate model. It reads: $m\ddot{x}(t) = -\nu\dot{x}(t) - kx(t)$.

By writing this second order ODE as a first order system, we obtain:

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \end{bmatrix} = \mathbf{f}(\mathbf{x}(t)) = \begin{bmatrix} f_1(\mathbf{x}(t)) \\ f_2(\mathbf{x}(t)) \end{bmatrix} = \begin{bmatrix} y(t) \\ -\frac{k}{m}x(t) - \frac{\nu}{m}y(t) \end{bmatrix} \tag{6.3}$$

with $x_1 = x$ and $y \equiv x_2 = \dot{x}$ the mass velocity.

In the present example, the model term f_1 contains a single term; f_2 contains two terms.

All terms of f are first-order polynomials.

For larger displacements: the complete non linear model If the displacements are moderate or large, non-linear effects become non negligible: an additional third order term has to be introduced in the expression of $f(\mathbf{x})$. The equations writes as follows, see Fig. 6.2 for the notations:

$$m\ddot{x}(t) = -\nu\dot{x}(t) - k_1 x(t) - k_3 x^3(t) \tag{6.4}$$

This equation will be the one considered in the numerical experiments (see next Section).

Example #2: the pendulum dynamics

Let us consider a pendulum of mass m and length L.

Its dynamics equations reads: $mL\ddot{\theta}(t) = -mg\sin(\theta)$, with g the gravity constant.

This system re-writes as:

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \end{bmatrix} = \mathbf{f}(\mathbf{x}(t)) = \begin{bmatrix} f_1(\mathbf{x}(t)) \\ f_2(\mathbf{x}(t)) \end{bmatrix} = \begin{bmatrix} y(t) \\ -\frac{g}{L} sin(x(t)) \end{bmatrix} \tag{6.5}$$

with: $x_1 = x = \theta$ the rotation angle with respect to vertical, $y = x_2 = \omega$ the angular velocity.

The RHS term f_1 contains a single term; same for f_2.

In the present case, the RHS does not contain polynomial terms only; it contains the sin term too.

6.1.2 The inverse problem

The two examples above are very basic; however in real-life ODE models, the variety of terms constituting the RHS is quite limited too. In others words, a few terms only constitute the great variety of the employed ODE models: polyniomals terms and combination between each others, trigonometric terms etc

Then, *the basic idea is to define an a-priori dictionary which may contain the relevant terms* to represent the measured phenomena.
Next, *using a large amount of measurements (the dataset) and the compress sensing property of L^1-norm based optimisation formulations, one expect to identify the relevant terms among all terms present in the dictionary.*

Notations The matrix notations are detailed in Section **??**.

In all the sequel, the employed vector size indices are as follows:

- $m = 1, \cdots, M$, the time instant index,

- $n = 1, \cdots, N$, the state component index,
- $d = 1, \cdots, D$, the dictionary component index.

The a-priori defined dictionary D

We build up a dictionary containing all a-priori relevant terms in the RHS $\mathbf{f}$.

For $\mathbf{x} \in \mathbb{R}^N$, a typical dictionary may be:

$$\mathbf{D}(\mathbf{x}^T) = \begin{bmatrix} 1 & \mathbf{x}^T & (\mathbf{x}^T)^2 & (\mathbf{x}^T)^3 & \cdots & \sin(\mathbf{x}^T) & \cos(\mathbf{x}^T) & \sin(2\mathbf{x}^T) & \cdots \end{bmatrix} \in \mathbb{R}^{1\times D} \tag{6.6}$$

By convention, $\mathbf{D}$ is a line vector. $(\mathbf{x}^T)^2$ denotes the line vector containing all the quadratic possible terms:

$$(\mathbf{x}^T)^2 = \begin{bmatrix} x_1^2, & x_1x_2, & x_2^2, & x_1x_3, & \cdots & , x_N^2 \end{bmatrix} \tag{6.7}$$

Similarly, $(\mathbf{x}^T)^3$ denotes the line vector containing all the cubic possible terms; etc.

Also we have: $\sin(\mathbf{x}^T) = \begin{bmatrix} \sin(x_1), & \sin(x_2), & \cdots & , \sin(x_N) \end{bmatrix}$.

The ODE system re-formulated in function of D

Let us write the ODE system (6.1) from the dictionary $\mathbf{D}$ terms.

To do wo, we introduce the coefficients matrix $\mathbf{K} = \begin{bmatrix} \mathbf{k}_1 & , \mathbf{k}_2, & \cdots & , \mathbf{k}_N \end{bmatrix} \in \mathbb{R}^{D\times N}$ such that:

$$\dot{\mathbf{x}}^T(t) = \mathbf{D}(\mathbf{x}^T(t))\mathbf{K}, \quad t \in [0, T] \tag{6.8}$$

Let us denote by $\mathbf{k}_n$, $n = 1, 2, \cdots, N$, $\mathbf{k}_n \in \mathbb{R}^D$, the column vector corresponding to the derivative of x_n in the ¡¡basis¿¿ $\mathbf{D}(\mathbf{x}^T)$. We obtain:

$$\dot{x}_n(t) = \mathbf{D}(\mathbf{x}^T(t))\mathbf{k}_n, \quad t \in [0, T] \tag{6.9}$$

The equation of (6.8) is equivalent to (6.1) with:

$$\mathbf{f}(\mathbf{x}) = \mathbf{K}^T \left(\mathbf{D}(\mathbf{x}^T)\right)^T, \quad \mathbf{x} \in \mathbb{R}^N \tag{6.10}$$

Recall that: $\mathbf{f} : \mathbb{R}^N \mapsto \mathbb{R}^N$.

Of course, it is expected (and assumed) that all terms of $\mathbf{f}$ are present in the dictionary $\mathbf{D}$.

Remark 6.1. *The components of* $\mathbf{D}$ *must be linearly independent to well define the coefficient matrix* $\mathbf{K}$*. They don't have a-priori to be orthogonal; however, if orthogonal, the results may be better, see later.*

The use of the transpose operator ^{T} is necessary when introducing the (discrete) matrices, in particular to adopt the convention that columns represent the field component in time instant n.

The discrete fields and system

Time discretization Let $\mathbf{t}_M = [t_1, t_2, \cdots, t_M]^T$ be the discrete time instants set with: $t_0 = 0, t_M = T$.

The discrete state matrix X We set:

$$\mathbf{X} = \mathbf{x}^T(\mathbf{t}_M) = \begin{bmatrix} \mathbf{x}^T(t_1) \\ \mathbf{x}^T(t_2) \\ \vdots \\ \mathbf{x}^T(t_M) \end{bmatrix} = \begin{bmatrix} x_1(t_1) & x_2(t_1) & \cdots & x_N(t_1) \\ x_1(t_2) & x_2(t_2) & \cdots & x_N(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ x_1(t_M) & x_2(t_M) & \cdots & x_N(t_M) \end{bmatrix} \in \mathbb{R}^{M \times N} \tag{6.11}$$

The lines of $\mathbf{X}$ correspond to the time instant; the columns correspond to the different state components $x_1,\ x_2, \cdots, x_N$.

The discrete derivatives matrix $\dot{X}$ We set:

$$\dot{\mathbf{X}} = \dot{\mathbf{x}}^T(\mathbf{t}_M) = \begin{bmatrix} \dot{\mathbf{x}}^T(t_1) \\ \dot{\mathbf{x}}^T(t_2) \\ \vdots \\ \dot{\mathbf{x}}^T(t_M) \end{bmatrix} = \begin{bmatrix} \dot{x}_1(t_1) & \dot{x}_2(t_1) & \cdots & \dot{x}_N(t_1) \\ \dot{x}_1(t_2) & \dot{x}_2(t_2) & \cdots & \dot{x}_N(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ \dot{x}_1(t_M) & \dot{x}_2(t_M) & \cdots & \dot{x}_N(t_M) \end{bmatrix} \in \mathbb{R}^{M \times N} \tag{6.12}$$

The discrete dictionary

Let us build up the dictionary at all instants $t_1, t_2, \cdots, t_M$ as follows:

$$\mathbf{D}(\mathbf{X}) = \begin{bmatrix} \mathbf{D}\left(\mathbf{x}^T(t_1)\right) \\ \vdots \\ \mathbf{D}\left(\mathbf{x}^T(t_M)\right) \end{bmatrix} = \begin{bmatrix} | & | & | & | & & | & \\ 1 & \mathbf{X} & \mathbf{X}^2 & \mathbf{X}^3 & \cdots & \sin(\mathbf{X}) & \cdots \\ | & | & | & | & & | & \end{bmatrix} \in \mathbb{R}^{M \times D} \tag{6.13}$$

$D(X)$ is a matrix $M \times D$.

For example, the third column reads:

$$\mathbf{X}^2 = \begin{bmatrix} (\mathbf{x}^T(t_1))^2 \\ (\mathbf{x}^T(t_2))^2 \\ \vdots \\ (\mathbf{x}^T(t_M))^2 \end{bmatrix} = \begin{bmatrix} x_1^2(t_1) & x_1(t_1)x_2(t_1) & x_2^2(t_1) & \cdots & x_N^2(t_1) \\ x_1^2(t_2) & x_1(t_2)x_2(t_2) & x_2^2(t_2) & \cdots & x_N^2(t_2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_1^2(t_M) & x_1(t_M)x_2(t_M) & x_2^2(t_M) & \cdots & x_N^2(t_M) \end{bmatrix}$$

The discrete system The discrete system corresponding to (6.8) follows:

$$\dot{x}_n(t_m) = \mathbf{D}\big(\mathbf{x}^T(t_m)\big)\mathbf{k}_n \quad \text{for } m = 1, \cdots, M, \text{ and } \; n = 1, \cdots, N \tag{6.14}$$

In matrix form, it reads:

$$\boxed{\dot{\mathbf{X}} = \mathbf{D}\big(\mathbf{X}\big)\mathbf{K} \quad \text{in } \mathbb{R}^{M\times N}} \tag{6.15}$$

The measurement matrices

Let us introduce the measurements matrices.

Observation of the state $\mathbf{x}(t)$ As already mentioned, we assume that the state X has been measured at all instants $t_1, t_2, \cdots, t_M$.

Then, we define $\mathbf{X}^{\text{obs}}$ as:

$$\mathbf{X}^{\text{obs}} = \begin{bmatrix} \mathbf{x}_1^{\text{obs}} & \mathbf{x}_2^{\text{obs}} & \cdots & \mathbf{x}_N^{\text{obs}} \end{bmatrix} = \begin{bmatrix} x_{11}^{\text{obs}} & x_{12}^{\text{obs}} & \cdots & x_{1N}^{\text{obs}} \\ x_{21}^{\text{obs}} & x_{22}^{\text{obs}} & \cdots & x_{2N}^{\text{obs}} \\ \vdots & \vdots & \ddots & \vdots \\ x_{M1}^{\text{obs}} & x_{M2}^{\text{obs}} & \cdots & x_{MN}^{\text{obs}} \end{bmatrix} \in \mathbb{R}^{M\times N} \tag{6.16}$$

The scalar value x_{mn}^{obs} denotes the observation of the (exact) state value $x_n(t_m)$, for $m = 1, \cdots, M$ and $n = 1, \cdots, N$.

The vectorial value $\mathbf{x}_n^{\text{obs}}$ (and matrix $\mathbf{X}^{\text{obs}}$) correspond to the discrete version of the exact vectorial value $x_n(\mathbf{t}_M)$ (and matrix $\mathbf{X}$ respectively).

Observation of the state derivatives $\dot{\mathbf{x}}(t)$ Similarly, we define:

$$\dot{\mathbf{X}}^{\text{obs}} = \begin{bmatrix} \dot{\mathbf{x}}_1^{\text{obs}} & \dot{\mathbf{x}}_2^{\text{obs}} & \cdots & \dot{\mathbf{x}}_N^{\text{obs}} \end{bmatrix} = \begin{bmatrix} \dot{x}_{11}^{\text{obs}} & \dot{x}_{12}^{\text{obs}} & \cdots & \dot{x}_{1N}^{\text{obs}} \\ \dot{x}_{21}^{\text{obs}} & \dot{x}_{22}^{\text{obs}} & \cdots & \dot{x}_{2N}^{\text{obs}} \\ \vdots & \vdots & \ddots & \vdots \\ \dot{x}_{M1}^{\text{obs}} & \dot{x}_{M2}^{\text{obs}} & \cdots & \dot{x}_{MN}^{\text{obs}} \end{bmatrix} \in \mathbb{R}^{M\times N} \tag{6.17}$$

Here, the scalar value $\dot{x}_{mn}^{\text{obs}}$ denotes the observation of the (exact) state value $\dot{x}_n(t_m)$, for $m = 1, \cdots, M$ and $n = 1, \cdots, N$.

Observations frequency vs model dynamics frequency *

We have assumed that one has measured the quantity $x_n(t)$ at the instants $t_1, \cdots, t_M$. This implicitly implies that one has measured the phenomena dynamics at the aforementionned time scale.

Moreover, the method requires to measure the derivatives of $x_n(t)$ too. Therefore this may be a serious limitating feature of the method...

even if one can (roughly) estimate $\dot{x}^{\text{obs}}_{mn}$ by Finite Differences from the values x^{obs}_{mn}.

Moreover, one has assumed above that the time sampling $\{t_1, \cdots, t_M\}$ correspond to the time numerical scheme too.

This implictly means that one has assumed that one are able to observe the system at the numerical time grid scale.

For real-life problems, the time grid of measurements are greatly larger than the required computational grid.

Indeed, one generally have sparse, relatively low frequent and noisy measurements of the modelled phenomena.

The accuracy of measurements for the derivatives may be very rough and are relatively accurate at large scale only. The latter are generally sensitive to the measurements errors.

In summary, the considered time sampling $\{t_1, \cdots, t_M\}$ represent the observed time scale. And in the present description, the latter has to be fine enough to provide stable and accurate numerical scheme too...

6.1.3 The optimisation problem

The convex non differentiable optimization problems

Given the observations matrix X^{obs} containing the state measurements and its derivatives at all instants, the model learning formulation may be written as follows.

Solve the N following (non differentiable) optimization problems:

$$(P_n) \; : \quad \min_{\mathbf{k}_n \in \mathbb{R}^D} \left\{ \|\mathbf{D}\big(\mathbf{X}^{\text{obs}}\big)\mathbf{k}_n - \dot{\mathbf{x}}^{\text{obs}}_n\|_2^2 + \lambda \|\mathbf{k}_n\|_1 \right\} \qquad \text{for } n = 1, \cdots, N \tag{6.18}$$

with $\lambda > 0$ a weight parameter.

Recall that $[\mathbf{k}_n]_{n=1,\cdots,N} = \mathbf{K}$ denotes the coefficients of the derivatives $[\dot{x}_n]_{n=1,\cdots,N} = \dot{\mathbf{x}}^T$ in the dictionary $\mathbf{D}$.

$\forall n = 1, \cdots, N$,

$$\mathbf{k}_n \in \mathbb{R}^D, \mathbf{X}^{\text{obs}} = \begin{bmatrix} \mathbf{x}_1^{\text{obs}} & \cdots & \mathbf{x}_N^{\text{obs}} \end{bmatrix} \in \mathbb{R}^{M\times N} \text{ and } \mathbf{D}(\mathbf{X}^{\text{obs}}) \in \mathbb{R}^{M\times D} \tag{6.19}$$

with N the number of states, M the number of instants, D the number of terms in the dictionary $\mathbf{D}$.

In matricial form, the N problems (P_n) can be reformulated as follows.

Find the matrix $\mathbf{K}$, solution of:

$$(P) : \min_{\mathbf{K}\in\mathbb{R}^{D\times N}} \left\{ \|\mathbf{D}\big(\mathbf{X}^{\text{obs}}\big)\mathbf{K} - \dot{\mathbf{X}}^{\text{obs}}\|_2^2 + (\lambda N)\, \|\mathbf{K}\|_1 \right\} \tag{6.20}$$

(To obtain the implication between the N problems (P_n) and Problem (P), the weight parameter has to be equal to (λN)).

The matrix norm definitions are recalled in Section **??**.

Recall that the L^1-norm is the ¡¡closest convex relaxation¿¿ of the pseudo-norm l^0,

$$\|\mathbf{k}\|_0 = \#\{k_d : k_d \neq 0,\ d = 1, \cdots, D\} = \sum_{d=1}^{D} \mathbf{1}_{k_d\neq 0} \tag{6.21}$$

Note that if setting the minimisation problem as a simple least-square problem i.e. in the L^2-norm only, then the optimal solution $\mathbf{K}$ would be optimal in the least-square sense given the a-priori defined *entire* dictionary $\mathbf{D}$. This would not enable to identify the correct model terms. The L^1-norm with its compress sensing properties does.

The present model learning method is illustrated in Fig. 6.1.

An identifiability condition

It has been demonstrated in the 2000's that a ¡¡sparse¿¿ signal k (that is containing a great majority of vanishing coefficients) can be identified from an amount of measurements much lower than for a ¡¡complete¿¿ signal. Let us apply the estimation presented in [13] in the present context.

Let us consider $\mathbf{k} \in \mathbb{R}^D$, k is supposed to be sparse. Let $\mathbf{D} \in \mathbb{R}^{M\times D}$ be an *orthonormal* matrix. We consider the measurements $\mathbf{y^{obs}} = \mathbf{Dk}$, $\mathbf{y} \in \mathbb{R}^M$.

In the worse case, the number of measurements M has to satisfy $M \approx D\log(D)$; while in the best case, $M \approx \log(D)$.

indeed, the following identifiability condition holds, see [13]:

$$M \geq \text{cst}\, \mu^2(\mathbf{D})\, S\, \log(D) \tag{6.22}$$

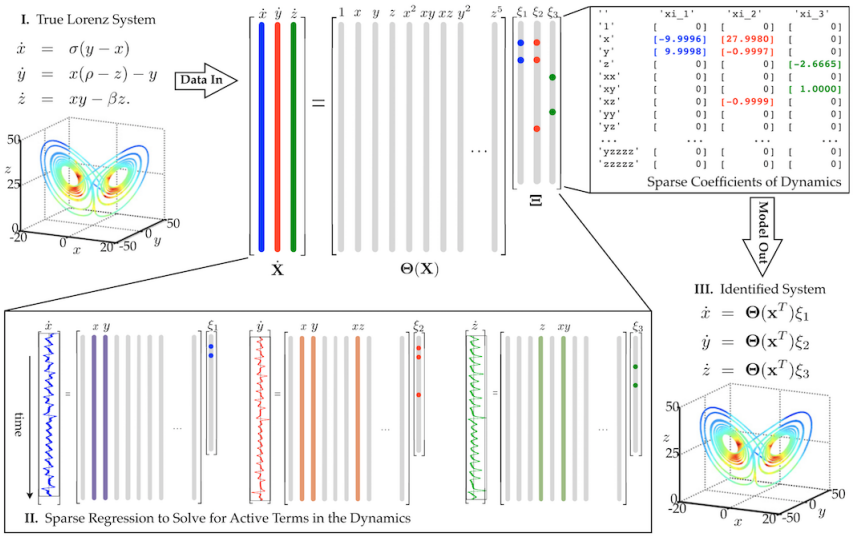


Figure 6.1: The model learning method as presented in [12]. Illustration based on the (chaotic) Lorenz dynamical system.

with the constant $cst \approx 1$, $\mu(\mathbf{D}) = \sqrt{P}\max_{m,d}(|D_{md}|)$ a conditionning measure of $\mathbf{D}$ ($\mu(\mathbf{D}) \approx 1$ if $\mathbf{D}$ orthonormal) and S the non vanishing component number of $\mathbf{k}$.

With inaccurate, noisy measurements

In real-life problems, even if the measurements of the state $\mathbf{X}$ are very accurate, the measurements of its derivatives $\dot{\mathbf{X}}$ are likely not, see the discussion above. These derivatives may be derived from the measurements of the state.

Therefore in practice one has to consider the following perturbed system:

$$\dot{\mathbf{X}}^{\text{obs}} = \mathbf{D}\big(\mathbf{X}^{\text{obs}}\big)\mathbf{K} + \eta\mathbf{Z} \tag{6.23}$$

with $\mathbf{Z}$ an aleatory variable following e.g. the normal distribution, and η the standard deviation value.

This uncertainty introduces an additional (aleatory) RHS in the ODE system.

Case ¡¡the state derivatives cannot be measured¿¿: towards an optimal control problem...*

The method previously presented requires to have measured both the state $\mathbf{X}$ and its derivative $\dot{\mathbf{X}}$ at all time steps, equivalently at a given time scale, cf the previous discussion.

In the case one can measure the state values $\mathbf{X}^{\text{obs}}$ at time instants $\mathbf{t}_M = [t_1, \cdots, t_M]^T$ not frequently enough to estimate satisfactorily the derivatives $\dot{\mathbf{X}}$ at the same time scale, the method

re-writes as follows.

Given a dictionary $\mathbf{D}$, given the initial state $\mathbf{x}_0 \in \mathbb{R}^N$, given the observations instants $\mathbf{t}_M = [t_1, \cdots, t_M]^T \in \mathbb{R}^M$, given the observations matrix $\mathbf{X}^{\text{obs}} \in \mathbb{R}^{M \times N}$,

find the matrix coefficients $\mathbf{K}$ satisfying:

$$\min_{\mathbf{K} \in \mathbb{R}^{M \times D}} \left\{ \|\mathbf{X}_\mathbf{K} - \mathbf{X}^{\text{obs}}\|_2^2 + \lambda \|\mathbf{K}\|_1 \right\} \tag{6.24}$$

$$\text{under the contraints } \begin{cases} \dot{\mathbf{x}}_\mathbf{K}^T(t) = \mathbf{D}(\mathbf{x}_\mathbf{K}^T(t))\mathbf{K}, \ t \in [0, T] \\ \mathbf{x}_\mathbf{K}(0) = \mathbf{x}_0 \\ \mathbf{X}_\mathbf{K} = \mathbf{x}_\mathbf{K}^T(\mathbf{t}_M) \end{cases} \tag{6.25}$$

This is an optimal control problem where $\mathbf{K}$ is the control variable.

Remark 6.2. *In the present case, the ¡¡problem components¿¿ are not independent of each others. Indeed, changing* $\mathbf{k}_n \in \mathbf{K}$*, implies a change of all others components through the model. This is not the case in Problem* (6.18)*. Indeed in this last case, each vector* $\mathbf{k}_n, \ n = 1, \cdots, N$ *can be computed independently of each other.*

6.1.4 The minimisation algorithms: LASSO, STRidge

To solve convex non differentiable optimization problem such as (6.18), the historical (and still efficient) algorithm is the Least Absolute Shrinkage and Selection Operator (LASSO) algorithm, [40], see Section **??**.

Among other possible algorithms, let us mention the STRidge (Sequentially Thresholded Ridge regression) algorithm and the ElasticNet algorithm, [44], the Section **??**.

A Ridge regression consists to consider the L^2-norm in (6.18) instead of L^1. The obtained Least Square solution will present small coefficients (but a-priori non vanishing) for the wrong model terms present in the dictionary.

The STRidge (Sequentially Thresholded Ridge regression) version consists to set to 0 values lower than a threshold, therefore imposing parcimony.

ElasticNet combines these two approaches by considering the two norms L^2 and L^1.

These algorithms are recalled in Section **??**.

Remark 6.3. *As already mentioned in the optimal control sections, the weigh parameter* λ *(see Section* **??***) has two roles: it balances the optimisation regularization term. In the LASSO algorithm case, it tunes the ¡¡importancy of parcimony¿¿.*

6.1.5 Case the goal is to identify model parameters *

Let us consider the scalar ODE model:

$$\frac{d\mathbf{x}}{dt}(t) = \mathbf{f}\big(\nu(t); \mathbf{x}(t)\big),\ t \in [0, T] \tag{6.26}$$

with $\nu(t) = [\nu_1(t), \nu_2(t), \cdots]^T$ an unknown, uncertain parameter to be identifed.

Basic principle

Following the principle of the model learning method above, one can introduce in the dictionary combinations of terms with ν.

For example: $\nu(t)x_1(t)x_2^2(t)$, etc.

Obviously this make highly increase the dictionary dimension...

As a consequence, this approach seems a-priori possible for a very few number L of parameters, $\nu(t) = (\nu_1(t), \cdots, \nu_L(t))$, only.

Examples of reduction of a model parameter $\nu(t)$

In real-life modeling problems, one may have good a-priori on the parameters form. In this case, one may be able to reduce a time or space dependent parameter to a very few number of unknown parameters. For examples:

- A sinusoidal form with unknown frequency, phase and amplitude: $\nu(t) = \nu_0 \sin(\omega_0 t + \varphi_0)$. This reduces the time dependent parameter $\nu(t)$ to three parameters only.
- An exponentially decreasing parameter : $\nu(t) = \nu_0 e^{-a_0 t}$. Two parameters only in this case.

General parametrization of $\nu(t)$

More generally, given a function basis $(\psi_n)_{n=1,\cdots,N_\psi}$ with $N_\psi \ll M$, one may set:

$$\nu(t) = \sum_{n=1}^{N_\psi} \nu_n \psi_n(t) \tag{6.27}$$

In this case, one has N_ψ parameters ¡¡only¿¿.
However, in the dictionary, these N_ψ additionnal unknown parameters may multiply the total number of entries in $\mathbf{D}$ by N_ψ.

It would provide $\mathbf{D} \in \mathbb{R}^{M \times N_\psi D}$. This may be prohibitive...

6.2 Numerical experiments

Let us illustrate this model learning method in the case of a simple non-linear oscillator.

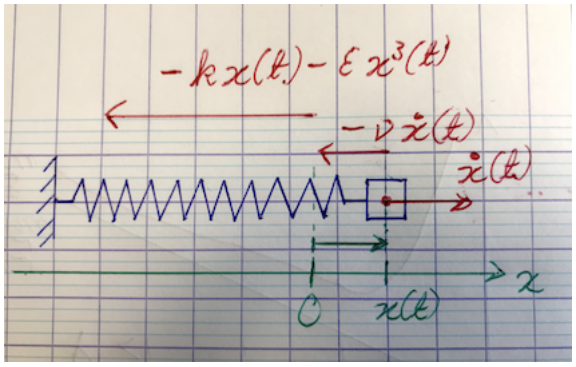


Figure 6.2: The spring system and notations. (Image extracted from S. Agdestein's INSA internship report).

6.2.1 The direct model: non-linear dynamics of a spring

The dimensionless equations

The considered model is the non linear spring dynamic equation (6.4). The state of the system in physical dimensions is denoted by $\tilde{x}\,[\mathrm{m}]$; it represents the mass displacement with respect to its equilibrium position, see Fig. 6.2. The final time is denoted by $\tilde{T}\,[\mathrm{s}]$.

The equation in physical dimensions reads:

$$m\ddot{\tilde{x}}(\tilde{t}) = -\tilde{\nu}\dot{\tilde{x}}(\tilde{t}) - \tilde{k}_1\tilde{x}(\tilde{t}) - \tilde{k}_3\tilde{x}^3(\tilde{t}),\ \tilde{t}\in[0,\tilde{T}] \tag{6.28}$$

with initial conditions.

The parameter m is the spring mass, $\tilde{\nu}\,[\frac{\mathrm{Ns}}{\mathrm{m}}]$ is a friction coefficient, $\tilde{k}_1\,[\frac{\mathrm{N}}{\mathrm{m}}]$ is a resistivity related coefficient, $\tilde{k}_3\,[\frac{\mathrm{N}}{\mathrm{m}^3}]$ is the actual resistivity coefficient.

The dimensionless system in its 1st order form Let us normalize and dimensionize the equations. The spring length is denoted by L.

We set: $\tilde{x}(\tilde{t}) = Lx(t)$, $\tilde{t} = T_0\tilde{t}$ with $T_0 = 2\pi\sqrt{\frac{m}{k_1}} = \frac{2\pi}{\omega_0}$. T_0 is the oscillation duration in the case of small displacements.

The dimensionless equation reads:

$$\dot{\mathbf{x}}(t) = \begin{bmatrix}\dot{x}(t)\\ \dot{y}(t)\end{bmatrix} = \mathbf{f}(\mathbf{x}(t)) = \begin{bmatrix}f_1(\mathbf{x}(t))\\ f_2(\mathbf{x}(t))\end{bmatrix} = \begin{bmatrix}y(t)\\ -\nu y(t) - k_1x(t) - k_3x^3(t)\end{bmatrix},\ t\in[0,T] \tag{6.29}$$

with $y\equiv\dot{x}$, and: $T = \frac{\tilde{T}}{T_0}$, $\nu = \frac{T_0}{m}\tilde{\nu} = \frac{2\pi\tilde{\nu}}{\sqrt{m\tilde{k}_1}}$, $k_1 = 4\pi^2$, $k_3 = \frac{L^2T_0^2}{m}\tilde{k}_3 = 4\pi^2L^2\frac{\tilde{k}_3}{\tilde{k}_1}$.

Note that smaller displacements are, lower the non linear terms is.

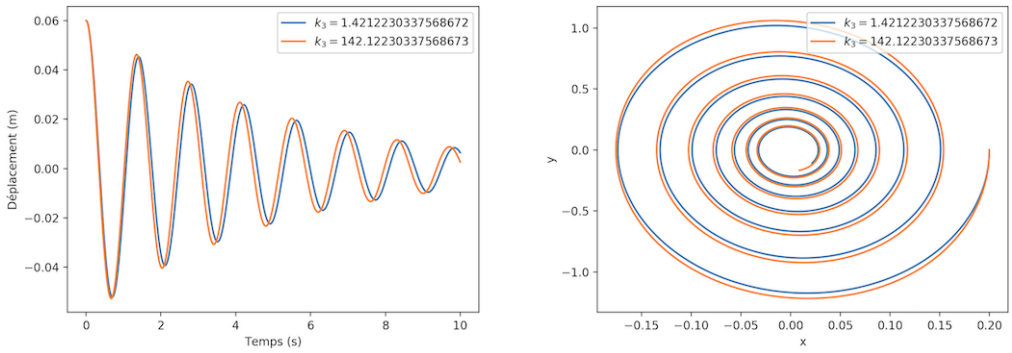


Figure 6.3: Direct simulation experiments: comparison of $x(t)$ vs time t (L) and the phase portrait (R), for two values of k_3 in the case of very small displacements (Case 1), see Tab. 6.1).

Preliminary simulations

The considered inverse problem is the following: *given measurements, do we manage to identify the dynamics equation terms ?*

Let us recall an evidence: before solving any inverse problem, one must perfectly understand the phenomena and the direct model !

Following this remark, let us perform few numerical direct simulations. Fig. 6.3 illustrates different direct simulations.

The dictionary definition

After having observed the system dynamics, typical questions a modeler may wonder are: is there non linear effects ? is there dissipation effects (friction terms) ?

Recall that a friction correspond a-priori to a third order term. Then, one select the a-priori potential functions:

$$\mathbf{D}(\mathbf{x}^T) = \begin{bmatrix} 1 & x & y & x^2 & xy & y^2 & x^3 & x^2y & xy^2 & y^3 \end{bmatrix} \in \mathbb{R}^{1\times 10} \tag{6.30}$$

In function of the dictionary $\mathbf{D}$, the direct model (6.29) reads as follows.

$$\dot{\mathbf{x}}^T(t) = \mathbf{D}(\mathbf{x}^T(t))\mathbf{K}^{\text{exact}}, \ t \in [0, T] \tag{6.31}$$

with $\mathbf{K}^{\text{exact}}$ defined by:

$$\mathbf{K}^{\text{exact}} = \begin{bmatrix} 0 & 0 \\ 0 & -k_1 \\ 1 & -\nu \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & -k_3 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{matrix} (1) \\ (x) \\ (y) \\ (x^2) \\ (xy) \\ (y^2) \\ (x^3) \\ (x^2y) \\ (xy^2) \\ (y^3) \end{matrix} \tag{6.32}$$

Therefore, one has the exact model which reads:

$$\begin{bmatrix} \dot{x} & \dot{y} \end{bmatrix} = \begin{bmatrix} 1 & x & y & x^2 & xy & y^2 & x^3 & x^2y & xy^2 & y^3 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & -k_1 \\ 1 & -\nu \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & -k_3 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{6.33}$$

The discrete system

The time sampling is here very fine: $\mathbf{t}_M = [t_m]_{m=1,\cdots,200}$, with $t_m = (m-1)\Delta t$ and $\Delta t = 0.05\,(\text{s})$. Therefore: $T = t_M = 10.0\,[\text{s}]$.

The dimension tab numbers are: $N = 2$, $M = 200$ and $D = 10$.

The discrete system reads: $\dot{\mathbf{X}} = \mathbf{D}\big(\mathbf{X}\big)\mathbf{K}$, with

$$\dot{\mathbf{X}} = \begin{bmatrix} \dot{x}(t_1) & \dot{y}(t_1) \\ \dot{x}(t_2) & \dot{y}(t_2) \\ \vdots & \vdots \\ \dot{x}(t_{200}) & \dot{y}(t_{200}) \end{bmatrix} \tag{6.34}$$

$$\mathbf{D}(\mathbf{X}) = \begin{bmatrix} 1 & x(t_1) & y(t_1) & x^2(t_1) & xy(t_1) & \cdots & y^3(t_1) \\ 1 & x(t_2) & y(t_2) & x^2(t_2) & xy(t_2) & \cdots & y^3(t_2) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x(t_{200}) & y(t_{200}) & x^2(t_{200}) & xy(t_{200}) & \cdots & y^3(t_{200}) \end{bmatrix} \tag{6.35}$$

k_1	$4\pi^2 \approx 39.478$
ν	0.562
k_3	1.421
T_0	7.118
Initial value x_0	Case 1: $x_0 = 0.030$ Case 2: $x_0 = 0.150$

Table 6.1: The parameters dimensionless values employed to generate the two measurements sets: the given parameters and the initial displacements values corresponding to the two cases. Recall that $L = 1$.

Synthetic measurements: twin experiments

Generation of the measurements To generate the measurements, the direct model is performed with some parameters values. This provides the measurements for the model term identification phase. Then the inverse problem conists to reciver the employed parameters values.

Such a procedure, based on synthetic measurements generated by the model, is called ¡¡twin experiments¿¿.

The employed parameter dimensionless values are indicated in Tab. 6.1.

Two set of measurements are generated. A first one with an initial displacement x_0 equal to 3% of L; a second one with x_0 equal to 15% of L.

The measurements of $x(t)$ are the resulting values of $x_1 = x$ at instants $t_1, \cdots, t_{200}$.

To define the measurements of $y = \dot{x}$ at these same instants, we compute the centered Finite Difference approximations.

Values of x and $\dot{x}$ constitutes the observation matrix $\mathbf{X}^{\text{obs}}$.

Next, to define the observation matrix $\dot{\mathbf{X}}^{\text{obs}}$, one approximates the second derivatives of x by finite differences again.

Remark. Measuring second derivatives of a field x in real-life phenomena may be highly critical, in particular at ¡¡small¿¿ scale.

It may be possible at larger scale than the one of x^{obs}.

This point is one of the limitation of the present model learning aproach.

Note that if performing the dimensionless dynamic system then the observations matrices have to be dimensionless too.

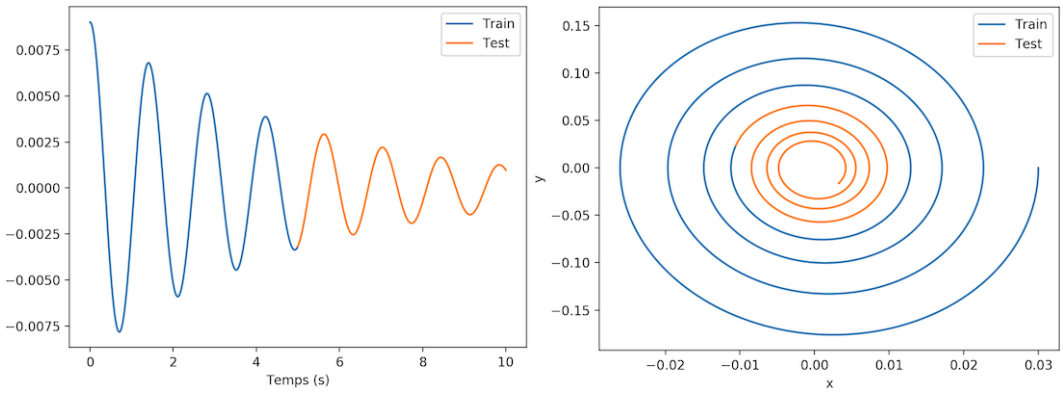


Figure 6.4: The train set ($[0, T/2]$) and the test set ($[T/2, T]$): (L) $x(t)$ vs time t. (R) The phase portrait.
¡¡Training¿¿ data are in blue (learning intervall); ¡¡testing¿¿-¿¿validation¿¿ data are in orange (validation intervall).

Learning set & testing set To evaluate the method, the measurements set are split in two sets:

- the first one containing the measurements from 0 to $T/2$, employed for the learning phase,

- the second one to compare the learned model prediction capabilities vs the true model dynamics, from $T/2$ to T.

Then, the matrices are split as follows:

$$\mathbf{D}\left(\mathbf{X}^{\text{obs}}\right) = \begin{bmatrix} \mathbf{D}\left(\mathbf{X}^{\text{obs.learn}}\right) & 50\% \\ \mathbf{D}\left(\mathbf{X}^{\text{obs.test}}\right) & 50\% \end{bmatrix} \tag{6.36}$$

$$\dot{\mathbf{X}}^{\text{obs}} = \begin{bmatrix} \dot{\mathbf{X}}^{\text{obs.learn}} & 50\% \\ \dot{\mathbf{X}}^{\text{obs.test}} & 50\% \end{bmatrix} \tag{6.37}$$

This decomposition is illustrated in Fig. 6.4.

6.2.2 The inverse problem

In this section we solve the inverse problem: to identify the model terms with their corresponding coefficients $k_\square$ and ν.

Given $(\mathbf{X}^{\text{obs.learn}}, \dot{\mathbf{X}}^{\text{obs.learn}})$ *and the dictionary* $\mathbf{D}$, *identify the dominating model terms among those present in* $\mathbf{D}$.

Recall of the optimization formulation

To solve this model learning problem, we solve the N convex non differentiable optimization problems 6.18 by using the LASSO algorithm, see Section **??** for details.

Recall the optimization problem to be solved:

$$\mathbf{K}^*(\lambda) = \underset{\mathbf{K}\in\mathbb{R}^{D\times N}}{\arg\min} \left\{ \|\mathbf{D}\big(\mathbf{X}^{\text{obs.learn}}\big)\mathbf{K} - \dot{\mathbf{X}}^{\text{obs.learn}}\|_2^2 + \lambda\|\mathbf{K}\|_1 \right\} \tag{6.38}$$

with $\lambda > 0$ a weight parameter to be a-priori set.

Setting the parsimony weight parameter λ using a peformance criteria

When solving this bi-objective optimization problem 6.18 (equivalently (6.38)), the setting of the weight parameter λ may be critical.

Larger λ is, more the compress sensing feature (i.e. $\|K\|_0$) is important (i.e. less model terms will be selected).

On the contrary, smaller λ is, less the compress sensing feature is important.

In practice, setting λ at correct values may be tricky.

The performance criteria to determine an optimal value of λ In the present twin experiments context, a good way to determine an optimal value of λ may be as follows.

- We define a *performance criteria* based on the test measurements set as follows:

$$J_\eta^{\text{test}}(\mathbf{K}) = \|\mathbf{D}\big(\mathbf{X}^{\text{obs.test}}\big)\mathbf{K} - \dot{\mathbf{X}}^{\text{obs.test}}\|_2 + \eta\ \|\mathbf{K}\|_0 \tag{6.39}$$

  The coefficient η is empirically set, see below.

- We solve the optimization problem (6.38) for numerous values of λ by starting with $\lambda = 1$. Next, λ is decreased untill the performance test (6.39) reaches a minimal value.

- For each obtained solution $\mathbf{K}^*(\lambda)$, the criteria $J_\eta^{\text{test}}(\mathbf{K}^*(\lambda))$ is calculated.
- Finally, the ¡¡optimal¿¿ value of λ is chosen as:

$$\lambda^* = \underset{\lambda>0}{\arg\min} \left\{ J_\eta^{\text{test}}(\mathbf{K}^*(\lambda)) \right\} \tag{6.40}$$

6.2.3 Numerical results

Analysis of the performance criteria

The figures 6.5 (Up) and 6.6 (Up) present the values of the performance criteria $J_\eta^{\text{test}}(\mathbf{K}^*(\lambda))$ vs λ.

In Fig. 6.5 (Top), $J_\eta^{\text{test}}(\mathbf{K}^*(\lambda))$ is plotted for three different values of η: $\eta = 10^{-4}\kappa(\mathbf{D})$, $10^{-3}\kappa(\mathbf{D})$ and $10^{-2}\kappa(\mathbf{D})$.

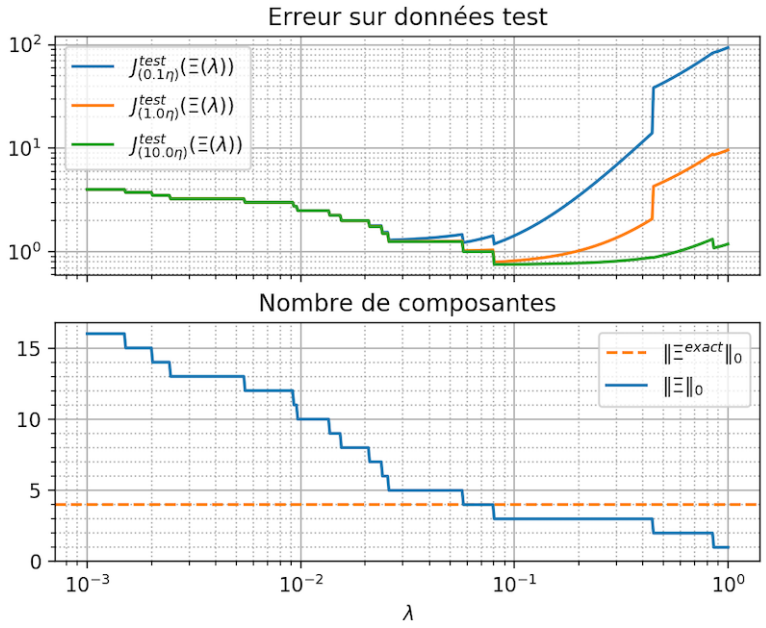


Figure 6.5: Case 1: $x_0 = 3\%$ of L (small displacements case). (Up) The performance criteria $J^{\text{test}}_{\eta}(\mathbf{K}^*(\lambda))$ vs λ for three values of η ($10^{-4}\kappa(\mathbf{D})$, $10^{-3}\kappa(\mathbf{D})$, $10^{-2}\kappa(\mathbf{D})$). We obtain: $\lambda^* \approx 0.08$. (Down) The argmin of $J^{\text{test}}_{\eta}(\mathbf{K}^*(\lambda))$ correspond well to the 3 non vanishing components of $\mathbf{K}$ (the non-linear term is too small to be influential).

Figures 6.5 and 6.6 differ from the initial displacements values x_0 only, see Tab. 6.1.

Identified model terms & results analysis

Let us recall the dimensionless coefficient matrix $\mathbf{K}$ value to be infered:

$$\mathbf{K}^{\text{exact}} = \begin{bmatrix} 0 & 0 \\ 0 & -39.478 \\ 1 & -0.562 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & -1.421 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{matrix} (1) \\ (x) \\ (y) \\ (x^2) \\ (xy) \\ (y^2) \\ (x^3) \\ (x^2y) \\ (xy^2) \\ (y^3) \end{matrix} \tag{6.41}$$

Therefore, one has to identify 4 non vanishing coefficients: 1 dominating (the linear term coefficient k_1) and 2 much smaller (the non linear terms coefficients k_3 and ν).

Case 1): very small displacements case As previously described, the numerical experiment start by setting $\lambda = 1$.

In this case, the algorithm makes vanish all coefficient excepted $(-k_1)$, see Fig. 6.5 for $\lambda = 1$. Its value is estimated at -31.211.

This estimation does not enable to accurately describe the spring dynamics.

For $\lambda = 0.5$, two non vanishing coefficients are identified, see Fig. 6.5 for $\lambda = 0.5$.

The obtained estimations are: $1 \approx 0.435$, $(-k_1) \approx (-35.038)$.

These values correspond to the dynamical system: $\dot{x} = 0.435y$ and $\dot{y} = -35.038x$.

By taking into account the I.C., this provides the solution: $x(t) = 0.03\cos(\sqrt{0.435 \times 35.038}t)$.

This cannot be a correct model since its solution $x(t)$ above does not damp as the observations do, see Fig. 6.4.

The performance criteria seems mininal for $\lambda = 0.08 \equiv \lambda^*$, with 3 identified terms, see Fig. 6.5.

For a greater value than λ^*, here $\lambda = 0.117$, the optimization algorithm provides the following estimations:

$$\mathbf{K}^*(\lambda = 0.117) = \begin{bmatrix} 0 & 0 \\ 0 & -39.479 \\ 0.868 & -0.562 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{matrix} (1) \\ (x) \\ (y) \\ (x^2) \\ (xy) \\ (y^2) \\ (x^3) \\ (x^2y) \\ (xy^2) \\ (y^3) \end{matrix} \quad \text{to be compared to } \mathbf{K}^{\mathbf{exact}} = \begin{bmatrix} 0 & 0 \\ 0 & -39.478 \\ 1 & -0.562 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & -1.421 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{matrix} (1) \\ (x) \\ (y) \\ (x^2) \\ (xy) \\ (y^2) \\ (x^3) \\ (x^2y) \\ (xy^2) \\ (y^3) \end{matrix} \tag{6.42}$$

The linear terms coefficients only are detected. The second column terms are accurately identified. The estimation of the coefficient 1 is still inaccurate.

For the half of the previous value, $\lambda = 0.055$ ($\lambda < \lambda^*$, see Fig. 6.5), all the terms including the non linear ones are accurately identified:

$$\mathbf{K}^*(\lambda = 0.055) = \begin{bmatrix} 0 & 0 \\ 0 & -39.478 \\ 0.938 & -0.562 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & -1.420 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{matrix} (1) \\ (x) \\ (y) \\ (x^2) \\ (xy) \\ (y^2) \\ (x^3) \\ (x^2y) \\ (xy^2) \\ (y^3) \end{matrix} \quad \text{to be compared to } \mathbf{K}^{\text{exact}} = \begin{bmatrix} 0 & 0 \\ 0 & -39.478 \\ 1 & -0.562 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & -1.421 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{matrix} (1) \\ (x) \\ (y) \\ (x^2) \\ (xy) \\ (y^2) \\ (x^3) \\ (x^2y) \\ (xy^2) \\ (y^3) \end{matrix} \tag{6.43}$$

For lower values of λ*,* more non vanishing coefficients are detected. However, in these cases the cost function is increasing, see Fig. 6.5. This is a typical overfitting behavior.

In conclusion, one easily detect the two dominant terms (the coefficients 1 and (-35.038)) but it is more tricky to detect the non linear terms (in the present very small displacements case...).

This result is consistent with the very small displacement dynamics. Indeed, in this case the non linear terms are negligible. Actually, non linear terms are almost useless in models for such small displacements.

Despite the challenging case, if setting the correct value for λ, we obtain an accurate estimation of all coefficients, including the non linear (small) terms.

Case 2): larger displacements case In this case and following the same procedure, one easily detect all coefficients, see Fig. 6.6.

Moreover, the estimated values of all coefficients are accurate, see (6.41) for the exact values. The obtained detailed results are not shown.

6.2.4 Programming practical

Upload the computational code (written in Python) available on the INSA Moodle plateform to make your own numerical experiments and analyse the results.

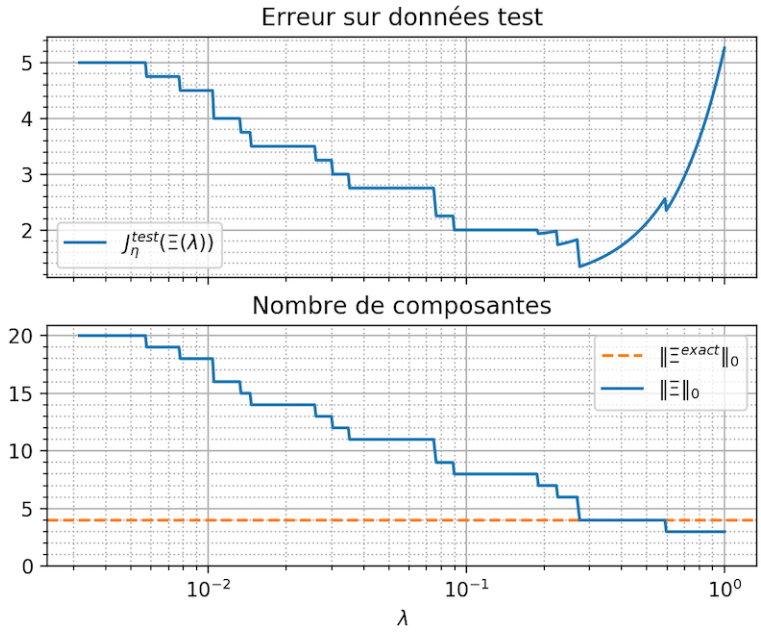


Figure 6.6: Same as Fig. 6.5 but in Case 2 i.e. for a larger initial displacement: $x_0 = 15\%$ of L. (Up) In this case and if considering J_η^{test} as performance criteria, we obtain $\lambda^* \approx 0.269$. (Down) The minimun of the performance criteria correspond to the 4 non vanishing components of $\mathbf{K}$ (the non-linear term is here detected).

6.3 Learning PDE model terms

Formally, the model learning method previously described can be extended to PDE models. This has first been done in [36].

6.3.1 Basic formalism for a generic scalar PDE equation

Let us consider the formal generic 2D scalar PDE equation:

$$u_t + A\left((\nu_1, \cdots, \nu_P); u, u_x, u_y, u_{xx}, u_{yy}, u_1 u_x, u_2 u_{y,\ldots}\right) = f(x,t) \text{ in } \Omega \times (0,T) \tag{6.44}$$

with $u(x,t)$ the unknown and $\nu = (\nu_1, \cdots, \nu_P)$ some model parameters.

Examples

The advection-diffusion equation In this formalism, the linear advective-diffusion equation reads:

$$u_t + A_{ad}((\nu, c_1, c_2); \partial_x u, \partial_y u, \partial^2_{xx} u, \partial^2_{yy} u) = f(x,t) \tag{6.45}$$

with:

$$A_{ad} = -\partial_x(\nu(x)\partial_x u(x,t)) - \partial_y(\nu(x)\partial_y u(x,t)) + c_1\partial_x u(x,t) + c_2\partial_y u(x,t) \tag{6.46}$$

with $c = (c_1,c_2)$ a given velocity field.

The viscous Burgers equation The viscous Burgers's equation corresponds to:

$$A_{ad} = -\partial_x(\nu(x)\partial_x u(x,t)) - \partial_y(\nu(x)\partial_y u(x,t)) + u_1\partial_x u(x,t) + u_2\partial_y u(x,t) \tag{6.47}$$

6.3.2 Example: the viscous Burgers equation

Dictionary definition

In the viscous 1D Burgers's equation case, the a-priori dictionary may be defined fo example as:

$$\mathbf{D}(u;\nu) = \left[1, \quad u, \quad u^2, \quad c_1 u\partial_x u \quad , \quad \cdots \quad , \quad \nu_P \partial^2_{xx} u\right] \tag{6.48}$$

Recall that if considering third order terms, waves dispersion may be captured.

By adopting the same formalism as in the ODE case, one has:

$$u_t = \mathbf{D}(u,\nu)\ \mathbf{K} \tag{6.49}$$

with $\mathbf{K}$ the matrix coefficient.

The discrete fields and system

On has to consider a time-space grid. In 1D, the most simple one is: $[(x_m,t_m)]_{m=1,\cdots,M}$.

Using this ¡¡square-based space-time grid¿¿, the discrete fields and matrices read:

$$\mathbf{U}_M = \begin{bmatrix} u(x_1,t_1) \\ \vdots \\ u(x_M,t_M) \end{bmatrix}$$

$$\mathbf{D}(\mathbf{U}_M,\nu_P) = \begin{bmatrix} \mathbf{D}(u(x_1,t_1),\nu_{1..P} \\ \vdots \\ \mathbf{D}(u(x_M,t_M),\nu_{1..P} \end{bmatrix}$$

The discrete system reads:

$$\mathbf{U}_t^M = \mathbf{D}(\mathbf{U}_M,\nu_P)\mathbf{K}$$

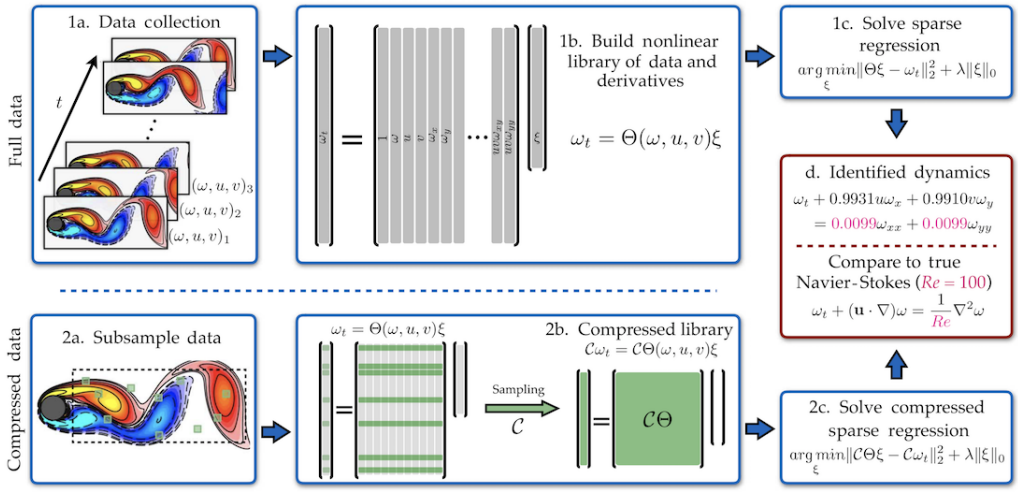


Figure 6.7: Illustration of the PDE model learning method from [36] with an application to the viscous Burger's equation.

Given $\mathbf{U}^{\text{obs}}$ the measurements of the field u, the method steps are as follows:

1. Compute by finite differences the derivatives of U^{obs},
2. Build up the dictionary $\mathbf{D}\big(\mathbf{U}, \partial_x\mathbf{U}_x, \partial^2_{xx}\mathbf{U}, \cdots\big) \in \mathbb{R}^{M\times D}$,
3. Solve the non differentiable optimization problem: $\mathbf{K} = \underset{\mathbf{K}\in\mathbb{R}^{D\times N}}{\arg\min}\{\|\mathbf{D}(\mathbf{U}, \cdots)\mathbf{K} - \mathbf{U}_t\|_2^2 + \lambda\ \|\mathbf{K}\|_1\}$.

Note that the present learning methods have been implemented in the Python PDE_FIND package, [12, 36]. The latter include various basic PDE examples.

Appendices

Appendix A

A few mathematical and optimization recalls

The outline of this chapter is as follows.

In this chapter, few basic mathematical and numerical concepts are shortly recalled. The reader is invited to consult his previous courses or courses available on-line for more details. We suggest the reading of the following excellent books: [1, ?] for recalls in Sobolev spaces, numerical analysis of PDE equations and differentiable optimization; also [11] for any recall or improvement in functional analysis.

A.1 Differential calculus, functional analysis

A.1.1 Differentials

First, let us recall the differentiability and differential definitions. In all the sequel, by default we denote by: X a Banach space, V a Hilbert space, $(.,.)$ its scalar product and $\|.\|$ its norm. The functional j is defined from X (or V) into $\mathbb{R}$. Ω denotes a bounded subset of $\mathbb{R}^d$; Ω is a Lipschitz domain (roughly, its boundary $\partial\Omega$ can be locally described as a Lipschitz graph).

The Fréchet-differential is defined as follows.

Definition A.1. *Let $u \in X$. The functional j is Fréchet-differentiable at point u, if there exists a linear form continuous L, $L \in \mathcal{L}(X, \mathbb{R}) = X'$, such that:*

$$j(u + \delta u) = j(u) + L(\delta u) + o(\delta u) \qquad with \lim_{\delta u \to 0} \frac{|o(\delta u)|}{\|\delta u\|} = 0$$

We call L the (Fréchet-)differential of j at point u, and we denote: $L = \dfrac{dj}{du}(u)$.

In all this manuscript, the differential $\frac{dj}{du}(u)$ can be denoted as follows too: $dj(u)$, $j'(u)$, depending on the context...

Next, we define the Gâteaux differential.

Definition A.2. *Let u and $\delta u \in X$. The functional j is Gâteaux-differentiable at point u in the direction δu, if there exists a linear form continuous L, $L \in \mathcal{L}(X, \mathbb{R}) = X'$, such that:*

$$\lim_{\varepsilon \to 0^+} \frac{1}{\varepsilon}[j(u + \varepsilon \delta u) - j(u)] = L(\delta u)$$

The Gâteaux-differentiable concept is weaker than the Fréchet-differential one ("Fréchet $\Longrightarrow$ Gâteaux" but the reciprocal is not true). Nevertheless its definition can be more convenient. In all the manuscript, when "differentiable" is mentioned, it means Fréchet-differentiable (hence Gâteaux-differentiable in any direction δu).

The definition extension to the second derivative is natural. Next the following Taylor expansion order 2 holds:

Proposition A.3. *If the functional j is twice differentiable then:*

$$j(u + \delta u) = j(u) + j'(u)(\delta u) + \frac{1}{2} j''(u)(\delta u, \delta u) + o(\|\delta u\|^2) \quad \text{with } \lim_{\delta u \to 0} \frac{o(\|\delta u\|^2)}{\|\delta u\|^2} = 0$$

where $j''(u)$ is a bilinear form defined from $X \times X$ into $\mathbb{R}$.

Exercice A.4. *Let X be a Banach space and j be a functional: $j : X \to \mathbb{R}$. - Calculate the differential expressions at the "point" u_0 in the direction δu, in the case:*
1) $j(u) = \int_\Omega u^p dx$, with $X = L^p(\Omega)$, $1 \leq p < \infty$.
- Calculate the 1st and 2nd order differential at u_0 in the direction δu and $(\delta u, \delta u)$ respectively, in the following cases:
2) $j(u) = \nabla u$, with $X = H^1(\Omega)$.
3) $j(u) = \int_\Omega \|\nabla u\|^2 dx$, X a subspace of $H^1(\Omega)$.

Exercice A.5. *Let V be a subspace of $H^1(\Omega)$ and let $a(.,.)$ be a bilinear form defined from $V \times V$ into $\mathbb{R}$. Give the differential expression $\partial_u a(u_0, v) \cdot \delta u$, in the case:*
1) $a(u, v) = \int_\Omega u^p v dx$, $1 \leq p < \infty$.
2) $a(u, v) = \int_\Omega \nabla u \nabla v dx$.

Exercice A.6. *Let V be a Hilbert space, let $a(.,.)$ be a bilinear form defined from $V \times V$ into $\mathbb{R}$, l a linear form from V into $\mathbb{R}$. Let j be the functional defined by: $j(u) = \frac{1}{2} a(u, u) - l(u)$. Prove that j is twice differentiable in V, and*

$$j'(u)(v) \equiv j'(u) \cdot v = a(u, v) - l(v) \quad \forall u \in V, \ \forall v \in V$$

$$j''(u)(v, w) = a(v, w) \quad \forall u, v, w \in V$$

Exercice A.7. *Let $V = L^2(\Omega)$, $a(.,.)$ the bilinear symmetric form defined on $V \times V$ by: $a(u,v) = \int_\Omega uvdx$ and $l(.)$ the linear form defined by: $l(v) = \int_\Omega fvdx$, with $f \in L^2(\Omega)$.*
We set: $j(u) = \frac{1}{2}a(u,u) - l(u)$.
By identifying V and V', prove that: $j'(u) = u - f$.

A.1.2 Green formulas

The formulas are key tools in the derivation of weak formulations of the model equations. Let Ω be a bounded subset of $\mathbb{R}^d$, Ω regular in the sense C^1 (roughly, its boundary $\partial\Omega$ is locally the graph of a C^1 function). Let $n = (n_i)_{1\leq i\leq d}$ be the external unit normal of the boundary. In Ω, dx denotes the volumic measure (Lebesgue's measure in dimension d), and ds the surface measure (Lebesgue's measure in dimension $(d-1)$ on the subvariety $\partial\Omega$).

We have the following important equalities.

Theorem A.8. *(Green formulae). Let Ω be a bounded subset of $\mathbb{R}^d$, C^1-regular, let v be a function in $C^1(\bar{\Omega})$ (its support is bounded in $\bar{\Omega}$). We have:*

$$\int_\Omega \frac{\partial v}{\partial x_i}(x)dx \;=\; \int_{\partial\Omega} v(x)n_i ds \quad 1 \leq i \leq d$$

Corollary A.9. *(Integration by part / Green formula). Let Ω be a bounded subset of $\mathbb{R}^d$, C^1-regular, let u and v be two functions in $C^1(\bar{\Omega})$ (with their support bounded in $\bar{\Omega}$). We have:*

$$\int_\Omega u(x)\frac{\partial v}{\partial x_i}(x)dx \;=\; -\int_\Omega \frac{\partial u}{\partial x_i}(x)v(x)dx + \int_{\partial\Omega} u(x)v(x)n_i ds \quad 1 \leq i \leq d$$

Corollary A.10. *Let Ω be a bounded subset of $\mathbb{R}^d$, C^1-regular, let u be a function in $C^2(\bar{\Omega})$, and v be a function in $C^1(\bar{\Omega})$ (with their support bounded in $\bar{\Omega}$). We have:*

$$\int_\Omega \Delta u(x)v(x)dx \;=\; -\int_\Omega \nabla u(x)\nabla v(x)dx + \int_{\partial\Omega} \frac{\partial u}{\partial n}(x)v(x)ds \quad 1 \leq i \leq d$$

with: $\frac{\partial u}{\partial n} = \nabla u \cdot n$.

A.1.3 The Riesz-Fréchet theorem

The Riesz-Fréchet representation theorem makes a connection between a Hilbert space and its dual space. Its result is useful to derive rigorously the adjoint operator including non trivial boundary conditions.

Theorem A.11. *(Riesz-Fréchet). Let V be a Hilbert space and L be an element of V'. It exists an unique element $l \in V$ such that:*

$$L(v) \equiv < L, v >_{V'\times V} = (l,v) \;\forall v \in V$$

Furthermore: $\|l\|_V = \|L\|_{V'}$.

In other words, the dual space V' can be represented by elements of V and the scalar product $(.,.)$ of V. Furthermore this defines an isomorphism between V' and V.

Exercice A.12. *Let V be a subspace of $H^1(\Omega)$, let $l(.)$ be a bilinear form defined from V into $\mathbb{R}$, and let $a(.,.)$ be a bilinear form defined from $V \times V$ into $\mathbb{R}$.*
Apply the Riesz-Fréchet representation theorem to the following cases:
1) $l(v) = \int_\Omega fvdx$, $f \in L^2(\Omega)$.
2) $a(u,v) = \int_\Omega \nabla u \nabla v dx$.
3) $a(u,v) = \int_\Omega \lambda \nabla u \nabla v dx + \int_\Omega cu^3 v dx$, *with* λ *and* u *given in* $L^\infty(\Omega)$.

A.1.4 The adjoint operator

Let E and F be two Banach spaces, E' and F' their dual space respectively.
Let A be a *linear operator*, unbounded, defined from $D(A) \subset E$ into F; $D(A)$ dense into E (see e.g. [11] for recalls of these concepts). Then its *adjoint operator* A^* satisfies:

$$< v, Au >_{F',F} = < A^*v, u >_{E',E} \quad \forall u \in D(A) \subset E,\ \forall v \in D(A^*) \subset F'$$

In the case $E = V$ Hilbert space (hence a reflexive Banach space), $A : V \to V'$, A linear, we have $A^* : V \to V'$, linear, such that:

$$\boxed{< v, Au >_{V,V'} = < A^*v, u >_{V',V} \quad \forall u \in V,\ \forall v \in V} \tag{A.1}$$

Exercice A.13. *Let V be a subspace of $H^1(\Omega)$, let A be the operator defined from V into V' as follows:*

$$A(u) = -div(\lambda \nabla u) + cu$$

with λ and u given in $L^\infty(\Omega)$.

Give an expression of A^* (potentially in a weak form), in the following cases:
1) $V = H_0^1(\Omega)$.
2) $V = H^1_{\Gamma_0}(\Omega) = \{v \in H^1(\Omega),\ v = 0 \text{ on } \Gamma_0\}$, *with* $\partial\Omega = \Gamma_0 \cup \Gamma_1$.

Exercice A.14. *Let V be a subspace of $H^1(\Omega)$, let A be the operator defined from V into V' as follows:*

$$A(u) = -div(\lambda \nabla u) + w \cdot \nabla u$$

with λ given in $L^\infty(\Omega)$, and w a vector field given in $(L^\infty(\Omega))^d$.

Give an expression of A^* (potentially in a weak form), in the following cases:
1) $V = H_0^1(\Omega)$.
2) $V = H^1_{\Gamma_0}(\Omega) = \{v \in H^1(\Omega),\ v = 0 \text{ on } \Gamma_0\}$, *with* $\partial\Omega = \Gamma_0 \cup \Gamma_1$, $w \cdot n = 0$ *on* Γ_0, $w \cdot n < 0$ *on* Γ_1.

A.2 Differentiable optimization

This section is a short summary of the optimization chapter of [1]; we refer to this book for the proofs and more details.

A.2.1 Convexity

Proposition A.15. *Let V be a Hilbert space, let j be a differentiable functional defined from V into $\mathbb{R}$. The following assertions are equivalent:*

i) *j is convex in V,*

ii)
$$j(v) \geq j(u) + (j'(u), v-u) \quad \forall u, v \in V$$

iii)
$$(j'(u) - j'(v), u-v) \geq 0 \quad \forall u, v \in V$$

Exercice A.16. *Prove that if the functional j is twice-differentiable then j is convex if and only if:*
$$j"(u)(w,w) \geq 0$$

A.2.2 Optimality conditions

Let us recall the first order necessary conditions of optimality.

Let j be a functional defined from V to $\mathbb{R}$. The optimization problem to be solved reads:

$$\inf_{u \in K} j(u)$$

with K a closed subset of V.

The optimality conditions are more simple in the case *K convex.*

Case A: K convex.

Indeed, the proof of first order optimality conditions tests the optimality of a candidate u in the *admissible direction* $(v-u)$ as follows: $w = u + t(v-u)$, $w \in K$ with $t \in [0,1]$.

Theorem A.17. *(Euler's inequality). Let j be differentiable, and let K be convex.*
i) If u is a local minimum of j in K, then:

$$(j'(u), v-u) \geq 0 \quad \forall v \in K$$

ii) Furthermore, if j convex, then u is a global minimum. □

The Euler's inequality is a *necessary* condition of optimality. If in addition j convex, then it is a *necessary and sufficient* condition (result ii)).

Theorem A.18. *(Euler's equality). Let j be differentiable. If $K = V$ or if the local minimum u is in the interior of K, then the previous inequality reads:*

$$(j'(u), v) = 0 \quad \forall v \in K$$

□

Exercice A.19. *Let A be $p \times n$-matrix, and $b \in \mathbb{R}^p$. Prove that the following least-square problem:*

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|^2$$

has one and only solution; and give the corresponding Euler's equality condition.

Case B: equality constraints, K affine subspace.
We consider the set of constraints K defined by:

$$K = \{v \in V, F(v) = 0\}$$

with $F(v) = (F_1(v), ..., F_m(v))$ a mapping defined from V into $\mathbb{R}^m$. Then, the necessary optimality conditions make introduce the Lagrange multipliers and read as follows.

Theorem A.20. *(Lagrange multipliers). Let j be differentiable and let F be C^1 in K. The vectors $(F_i'(v))_{1 \leq i \leq m}$ are supposed to be linearly independent.*
Then, if u is a local minimum of j in K, it exists m reals $\lambda_1, ..., \lambda_m$, called Lagrange multipliers, such that:

$$j'(u) + \sum_{i=1}^{m} \lambda_i F_i'(u) = 0$$

□

A.2.3 Lagrangian and saddle-point

Let us define the *lagrangian* $\mathcal{L}$ defined from $V \times \mathbb{R}^m$ into $\mathbb{R}$ by:

$$\mathcal{L}(v, \mu) = j(v) + \sum_{i=1}^{m} \mu_i F_i(v) = j(v) + \mu.F(v)$$

If u is a local minimum of j in K, Theorem A.20 states that it exists $\lambda \in \mathbb{R}^m$ such that (u, λ) is a *stationnary point* of $\mathcal{L}$, i.e.

$$\partial_v \mathcal{L}(u, \lambda) = 0 \text{ and } \partial_\mu \mathcal{L}(u, \lambda) = 0$$

Indeed, we have:

$$\partial_\mu \mathcal{L}(u, \lambda) = F(u) = 0 \text{ if } u \in K$$

and

$$\partial_v \mathcal{L}(u,\lambda) = j'(u) + \lambda.F'(u) = 0$$

The pair (u,λ), local solution of the minimization problem with constraint equality, can be interpreted a *saddle point* of the Lagrangian $\mathcal{L}$, i.e. a point (u,λ) which satisfies locally in $K \times \mathbb{R}^m$:

$$\forall \mu \quad \mathcal{L}(u,\mu) \leq \mathcal{L}(u,\lambda) \leq \mathcal{L}(v,\lambda) \quad \forall v$$

The duality theorem, see [1], implies that under the assumption of Theorem A.20 and if j strictly convex, then (u,λ) satisfies:

$$\min_v \max_\mu \mathcal{L}(v,\mu) = \max_\mu \min_v \mathcal{L}(v,\mu)$$

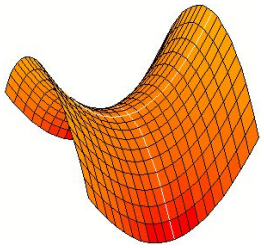


Figure A.1: A 2d saddle point (or a mountain pass) represented for a pair of scalar variables (v,μ)

For more details, we refer to [1].

Few extra crucial and classical exercices are proposed; all of them can be read in [1].

Exercice A.21. *(Quadratic optimization with linear constraints)*
Let A be a matrix order n, symmetric positive define. Let B be a rectangular matrix $m \times n$, let $b \in \mathbb{R}^n$. We seek to solve the optimization problem:

$$\inf_{x \in Ker(B)} j(x) \text{ with } j(x) = \frac{1}{2}(Ax,x) - (b,x)$$

1) Prove that it exists a solution if A positive; and the solution is unique if A positive define.
2) Prove that the unique solution x^* satisfies:

$$Ax^* - b = B^*p \text{ with } p \in \mathbb{R}^m$$

Exercice A.22. *Let $f \in L^2(\Omega)$, we consider the functional $j : V = H_0^1(\Omega) \to \mathbb{R}$ defined by:*

$$j(v) = \frac{1}{2}\int_\Omega \|\nabla v\|^2 dx - \int_\Omega f v dx$$

This functional represents for example the energy of an elastic membrane under the external force f, or the energy of any purely diffusive phenomena (eg a temperature, molecule diffusion etc).
We seek to solve:

$$\inf_{u \in V} j(u)$$

In others words, we seek the solution of the system at equilibrium (minimal energy).

1) Prove that the Euler equation gives the variational formulation:

$$\int_\Omega \nabla u \nabla v dx = \int_\Omega f v dx \quad \forall v \in H_0^1(\Omega)$$

(We remark that $H^1(\Omega)$ is the natural energy space for second order linear elliptic models).
2) Write the classical form of this equation.
3)
a) Prove that the energy functional is strictly convex in V.
b) After discretization (using FEM for example), what numerical algorithms would you use to solve this minimization problem ?
c) After discretization (using FEM), what algorithm would you use to solve the linear system corresponding to the variational formulation ?

A.3 Descent algorithms and Newton-like methods

For more details we refer to any excellent course on-line, or [1, 9].

The variational data assimilation method leads to the numerical minimization of a functional j; $j : V \to \mathbb{R}$, V Hilbert space. Thus, let us consider the following *discrete optimization problem*:

$$\text{Find } x^* \in \mathbb{R}^m \quad \text{such that:} \quad j(x^*) = \min_{x \in \mathbb{R}^m} j(x) \ ,$$

where the (discrete) cost function $j : \mathbb{R}^m \to \mathbb{R}$ is supposed to be C^1 (i.e. continuously differentiable).

If m is large (classical case in data assimilation), an affordable approach is to solve this minimization problem by using a *descent algorithm*.
Then the minimization is local only since if j is not convex, it gives at best *a local minimum*, on contrary to global minimization methods such as genetic algorithms for example.

A.3.1 Gradient type methods

A descent type algorithm is based on the *information of the gradient* of j ; that is why these algorithms require j to be C^1.

All gradient type methods can be formulated as follows:

$$\left\{ \begin{array}{ll} x^0 & \text{given} \\ x^{k+1} \; = \; x^k + \alpha^k d^k & \text{for } k = 0, \ldots \end{array} \right.$$

where d^k is the direction of descent, $d^k \in \mathbb{R}^m$, and α^k is the step of descent, $\alpha_k \in \mathbb{R}^+$.

By definition, d^k is a descent direction if and only if it satisfies: $< d^k, \nabla j(x^k) >< 0$.
Thus, it exists a scalar value α^k such that: $j(x^{k+1}) < j(x^k)$.

Exercice A.23. *Prove this assertion.*

Hint. Write a second order Taylor expansion: $j(x^{k+1}) = j(x^k) + \alpha^k < d^k, \nabla j(x^k) > +(\alpha^k)^2 < H_j^k d^k, H_j^k d^k > +O(\alpha^k \|d^k\|)$.

Then, defining a descent algorithm consists to define the direction d^k and the descent step α^k (linear search).

The steepest descent method. The most simple gradient method consists to "follow" the local steepest descent at each step i.e.: $d^k = -\nabla j(x^k)$.
This algorithm seems "natural", it is simple and it is interesting in view to illustrate descent algorithms in general. Nevertheless, it should never be used because absolutely inefficient.
It can be proved that if j is quadratic, the steepest descent method, associated with the so-called optimal step descent, converges. Nevertheless, at each iterate, one has: $< d^k, d^{k+1} >= 0$ i.e. a zig-zag behavior, see Fig. A.2.

The Conjugate-Gradient method. An improved version of the gradient method is the conjugate-gradient method. In the case j quadratic, it can be proved that CG algorithm converges with at most m iterates. Hence in theory, the CG algorithm is a direct method and not a iterative one. Nevertheless in practice this algorithm converges in only few iterations compared to m; that the reason why it is considered as an efficient and iterative method.

For a short but illustrative presentation for the CG algorithm, we refer to "An Introduction to the Conjugate Gradient Method Without the Agonizing pain, by Shewchuk (1994)" available on-line.

Remark A.24. *If j is strictly convex then CG algorithm is The right method to employ (typically j defined from a symmetrical positive define matrix).*
But if j is not convex (classical case in non-linear data assimilation problems), a quasi-Newton method should be prefered instead (see below).

Initial point. Let us point out that the choice of the initial point x^0 is very important, especially if j is not convex. As a matter of fact, it will determine the local minimum obtained.
In the 2d case ($m = 2$), you can mind to a small area in mountains; the algorithm is driving

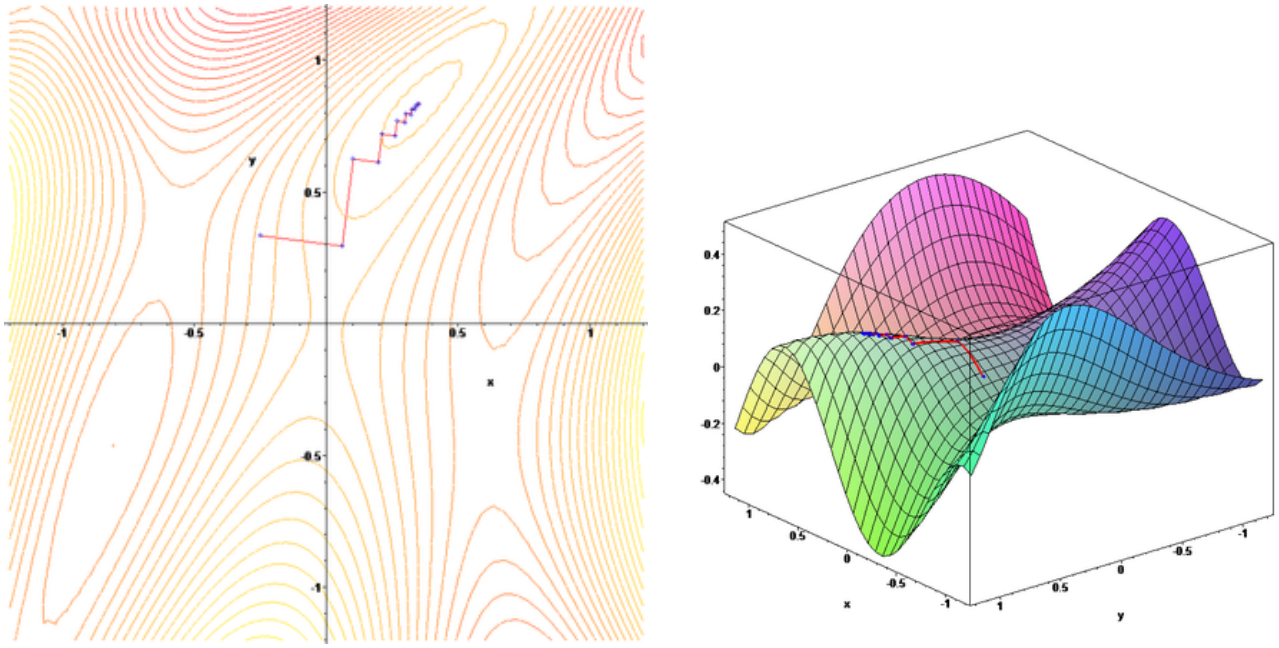


Figure A.2: The steepest descent algorithm with optimal step at each iterate, go down following a zig-zag route ! Courtesy of .

you to the nearest lake or valley bottom. Is that point you are looking for ?

A.3.2 The linear search (step descent)

Given d^k, computing the descent step is called the linear search (it is a 1d problem). An efficient method is based on the Wolf's conditions. The idea is to compute an 'acceptable' step length α^k that reduces the objective function 'sufficiently' (compared to its minimization in the direction d^k given). The Wolf's conditions (also called Armijo's rule) writes a follows. Determine α^k such that:

$$\begin{aligned} j(x^k + \alpha^k d^k) &\leq j(x^k) + \omega_1 \alpha^k \left\langle \nabla j(x^k), d^k \right\rangle \\ \left\langle \nabla j(x^k + \alpha^k d^k), d^k \right\rangle &\geq \omega_2 \left\langle \nabla j(x^k), d^k \right\rangle \end{aligned}$$

where ω_1 and ω_2 are two scalar values such that: $0 < \omega_1 < \frac{1}{2}$ and $\omega_1 < \omega_2 < 1$. Practical values can be for example: $\omega_1 = 10^{-4}$, $\omega_2 = 0.99$.

A.3.3 Newton method (2nd order)

Let us consider the first order necessary condition of minimum: $\nabla j(x) = 0$. Next, let us write the Newton's method (also called the Newton-Raphson method) to solve this non-linear system of equations in x.

We set: $G(x) = \nabla j(x)$. We obtain the Newton's iterates:

$$x^{k+1} = x^k - [DG(x^k)]^{-1} \cdot G(x^k)$$

Thus, the Newton's method can be viewed as a descent method with:

$$d^k = -\left[\nabla^2 j(x^k)\right]^{-1} \cdot \nabla j(x^k)$$

The Newton's method is very efficient if converging (since its quadratic convergence; recall j must be C^2).
The (potentially huge) drawback is the computation of the inverse of the Hessian $\left[\nabla^2 j(x^k)\right]^{-1}$ at reasonable cost...

For large dimensional problems (m large), the computation of the Hessian matrix is often not affordable. Then, two options are : Gauss-Newton method or Quasi-Newton method.

A.3.4 Gauss-Newton method

A first option is to solve the necessary condition by a Gauss-Newton algorithm (potentially incomplete). The Gauss-Newton method, used to solve non-linear least squares problems, can be seen as a modification of the Newton method to find a minimum. Unlike Newton's method, the Gauss-Newton algorithm has the advantage that second derivatives are not required. At the end, the algorithm consists to approximate: $\left[\nabla^2 j(x^k)\right]$ by $\left[\nabla j(x^k)\right]^T \left[\nabla j(x^k)\right]$.

Its convergence is not guaranteed, not even locally like it is in the Newton method case. The rate of convergence of the Gauss-Newton algorithm may "approach order 2". As usual, its convergence will depend on the choice of the initial guess x^0 and the matrix condition number.

A.3.5 Quasi-Newton method

A second option (the most classical) is to use a Quasi-Newton method. The basic idea of a Quasi-Newton algorithm is to approximate the inverse of the Hessian. Briefly, the descent direction d^k is computed as follows:

$$d^k = -W^k \cdot \nabla j(x^k) \tag{A.2}$$

where W^k is a matrix symmetric positive define which *approximates* the inverse Hessian $\left[\nabla^2 j(x^k)\right]^{-1}$, and using the gradient information only.
Thus, these methods remain first order methods and *the computation of the gradient only is required* (instead of the Hessian for Newton's method).

In some Quasi-Newton methods (not the one described below), d^k is computed by solving the linear system: $Z^k d^k = -\nabla j(x^k)$, with Z^k which approximates the Hessian $\left[\nabla^2 j(x^k)\right]$.

The most classical and efficient Quasi-Newton method is the so-called BFGS (for the names of its authors Broyden-Fletcher-Goldfarb-Shanno). The version so-called 'L-BFGS' (for Limited memory) is particularly efficient for m large. The BFGS algorithms have been implemented in all good computational librairies and software.
In the BFGS method, the matrices W^k are updated step-by-step using the formula BFGS inverse which requires the gradient only.
The matrix W^k is not stored in memory, but only couples of vectors which allow to rebuilt it. For more details, we refer the reader to [9].

An excellent implementation of L-BFGS algorithm in Fortran is the routine M1QN3 by C. Lemarechal, J.C. Gilbert from INRIA, France (see website dedicated, free program available on line). It is used for example by Meteo-France to compute its forecasts (4 times a day). It has been implemented into Scilab too.

Stopping criteria Few criteria can (must ?) be apply to stop the minimization process. First, one can test the (normalized) norm of the gradient:

$$\frac{\left\|\nabla j(x^k)\right\|}{\left\|\nabla j(x^0)\right\|} \leq \varepsilon \ .$$

with $\varepsilon \approx 10^{-5} - 10^{-7}$ typically.

Furthermore, since data assimilation can be employed *only with a good understanding of the physical system modeled* (it cannot be used reasonably as a black box...), a possible criteria is the stationarity either of the control variable and/or the cost function (both interpreted in a physical point of view). That is:

$$\frac{\left\|x^k - x^{k-1}\right\|}{\left\|x^k\right\|} \leq \varepsilon_1 \text{ and / or } \frac{|j(x^k) - j(x^{k-1})|}{|j(x^k)|} \leq \varepsilon_2$$

with ε_i defined by the end-user (with its own understanding of the physical system).

In addition, of course a maximum of iterate must be defined, in the case no 'convergence' is reached in a reasonable number of iterates.

A.4 Basic notations of statistics

Below very basic definitions (and notations) of probability and statistics are recalled.

Let $\hat{X}$ be an aleatory (random) variable.
Its average, also called the expected value, expectation or mean, is defined by:

$$\mu \equiv E[\hat{X}] = \int_a^b x f(x) dx$$

It is the probability-weighted average over all samples, $f(x)$ being the probability density function (pdf).

The covariance between two random variables $\hat{X}$ and $\hat{Y}$ is denoted by $cov(\hat{X}, \hat{Y})$ or $\sigma_{\hat{X}\hat{Y}}$. It measures how the two random variables change together.
If the two variables $\hat{X}$ and $\hat{Y}$ are independent then $cov(\hat{X}, \hat{Y}) = 0$. The reciprocal is not true. Random variables whose covariance is zero are called uncorrelated.
The definition is:

$$\sigma_{\hat{X}\hat{Y}} \equiv cov(\hat{X}, \hat{Y}) = E[(\hat{X} - E[\hat{X}])(\hat{Y} - E[\hat{Y}])]$$

We have the property:

$$cov(\hat{X}, \hat{Y}) = E[\hat{X}\hat{Y}] - E[\hat{X}]E[\hat{Y}]$$

The form $cov(.,.)$ is bilinear symmetrical positive define.
The associated quadratic form defines the variance. Its normalized version is the correlation coefficient $\rho(\hat{X}, \hat{Y})$:

$$\rho(\hat{X}, \hat{Y}) = \frac{\sigma_{\hat{X}\hat{Y}}}{\sigma_{\hat{X}}\sigma_{\hat{Y}}}$$

ρ measures the strength of the linear relation between $\hat{X}$ and $\hat{Y}$. It is a dimensionless number. If $\rho = 0$ the two variables are uncorrolated; if $\rho = \pm 1$, the two are linearly dependent.

The variance of a random variable or distribution, denoted by $Var(\hat{X})$ (or $\sigma^2_{\hat{X}}$), is a measure of how far the numbers lie from the average.
In other words, the variance is a measure of the amount of variation of values of the variable, taking account of all possible values and their probabilities or weightings (not just the extremes which give the range).
It is the expected value of the squared difference between the variable realisation and the variable mean:

$$\begin{aligned} \sigma^2_{\hat{X}} \equiv Var(\hat{X}) &= cov(\hat{X}, \hat{X}) \\ &= E[(\hat{X} - \mu)((\hat{X} - \mu)] \\ &= E[(\hat{X} - \mu)^2] \end{aligned}$$

The standard deviation ("ecart-type" in French) is the square root of the variance. It is denoted by $\sigma_{\hat{X}}$. It has the same dimension as the random variable $\hat{X}$.
Equivalently, it is the square root of the average value of $(\hat{X} - E(\hat{X})^2)$.

Roughly, the standard deviation measures the variation (dispersion) from the average. A low standard deviation indicates that data tend to be very close to the mean; high standard deviation indicates that data are spread out over a large range of values, Fig. A.3.

In statistics, an estimator is a rule to calculate the estimation of a given quantity from observed data.

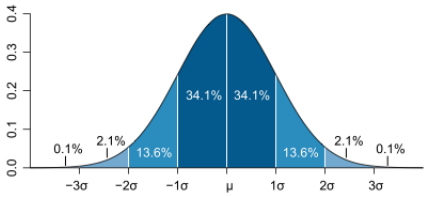


Figure A.3: A plot of a normal (Gaussian) distribution. Each band has a width of 1 standard deviation σ. Image from Wikipedia webpage.

The Mean Square Error (MSE) of an estimator quantifies the difference between values provided by the estimator and the true values.
The mean-square error measures how far the collection of values are in average from a estimated parameter x. It is defined by:

$$MSE(\hat{X}) = E((\hat{X} - x)^2)$$

We have:

$$MSE(\hat{X}) = Var(\hat{X}) + (Bias(\hat{X}, x))^2$$

Hence, if the random variable $\hat{X}$ is unbiased, the variance equals the MSE.

Appendix B

Algorithmic - Automatic Differentiation *

The outline of this chapter is as follows.

Contents

At the present stage, the reader known why it can be interesting to consider an adjoint model, what for. In practice, the strategy to write the adjoint model may be different according to the forward code structure. One of the three possibilities to derive an adjoint code consists to derive the forward *source code*. In C or Fortran it may be possible as a "source-to-source

transformation" process; in C++ by operators over-loading techniques.
In the present chapter, the forward source code is supposed to be written in Fortran (like many efficient computational codes).. In next section, the advantages and drawbacks of algorithmic-"automatic" differentiation is briefly presented. Next, we present the link between differential calculus (as employed in the course to derive the adjoint equation and the cost gradient) and a source code differentiation. The presentation is done in the framework of the automatic differentiation software Tapenade [23]. Moreover the conceptual way to re-employ your direct linear solver (eg based on your favourite linear algebra library) in the adjoint code; indeed the latter does not have to be differentiated. Finally, are presented some useful tricks for MPI instructions (see "how to adjointize the forward MPI instructions ?"), and how to save a bit of your memory. Indeed the memory will likely be your greatest issue (vs CPU time) while performing your "backward" adjoint code.
Finally let us remark thats the algorithmic differentiation described below to obtain the adjoint code (therefore the gradient) is similar to the so-called "backpropagation gradient" in the neural newtwork - deep learning community. More precisely, the "backpropagation gradient" is a particular case of the algorithmic differentiation (retropropagating by following the chain rule and saving in memory the intermediate states).

B.1 What adjoint code ?

In practice, there exists three approaches to compute the adjoint state variable, then the gradient of the cost function.

1. We write the continuous adjoint equations, we discretize them using an appropriate numerical scheme (a-priori the same numerical method than the direct model), we discretize the expression of the (continuous) gradient. We obtain the so-called *discretized continuous gradient*
2. We discretize the direct model. We derive these discrete equations in order to obtain the adjoint discrete equations. It is calculations in finite dimension spaces, that is why the present approach is the most famous one in engineering. We obtain the so-called *discrete gradient*.
3. The *computational gradient* is obtained from the differentiation of the forward code directly !

The three approaches are possible. The choice should be done upon the human time development required. The three gradients obtained are not exactly the same since propagation of all different errors is not the same for each approach. Very few numerical analysis results exits on the topic (e.g. in case of a linear PDE discretized using conforming finite element, the first two approaches are the same up to the scheme errors).
For non-linear and very large scale problems, some end-users claim that the tiny differences between the three gradients can justify differences of local convergence behaviours observed...

The algorithmic differentiation approach (last approach) may present two advantages. First, it may ensure a better consistency between the computed cost function, which is the output of

the direct code, and its gradient since it is the computed cost function which is differentiated (consistency including all types of errors: schemes, rounding errors, iterative algorithms etc). Second, a large part of this extensive task can be automated using algorithmic differentiation, see e.g. [20], if the direct code has been designed to it !... In case of a direct code written in Fortran, and initially if designed to be differentiated by an algorithmic process, the adjoint code can be almost automatically derived using an automatic differentiation software. One of the most efficient automatic differentiation tool, source-to-source, is Tapenade, see [23, 22].

The computational software DassFlow presented previously [17] has been initially design to be differentiated by Tapenade; that is why one can obtain very fast the adjoint code up-to-date. As a matter of fact, let us point out that as soon as the direct model is modified (extra terms, numerical approximation changed etc), the corresponding adjoint code must be modified in consequence, whatever the approach chosen.

B.2 From mathematical differentiation to source code differentiation

We describe how to define the direct code, then what is the response of the adjoint code automatically generated and finally how to use it. We follow the presentation done in [24], see [33] too.
Let $\mathcal{K}$ be the space of control variables and $\mathcal{Y}$ the space of the forward code response. In the case of DassFlow, we have :

$$\mathbf{k} = (y_0, q_{in}, z_{out}, n, z_b,)^T \quad \text{and} \quad Y = \big(y, j\big)^T$$

Let us point out that we include both the state and the cost function in the response of the forward code.

We can represent the direct code as the operator $\mathcal{M} : \mathcal{K} \longrightarrow \mathcal{Y}$, see figure B.1.

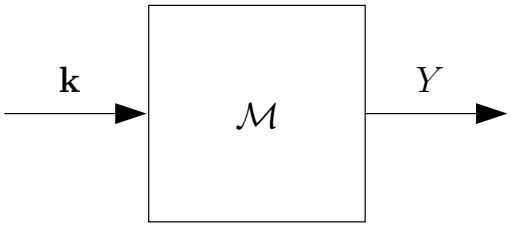


Figure B.1: Representation of the direct model.

The tangent model becomes $\frac{\partial \mathcal{M}}{\partial \mathbf{k}}(\mathbf{k}) : \mathcal{K} \longrightarrow \mathcal{Y}$. It takes as input variable a perturbation of the control vector $d\mathbf{k} \in \mathcal{K}$, then it gives the variation $dY \in \mathcal{Y}$ as output variable, see figure B.2:

$$dY = \frac{\partial \mathcal{M}}{\partial \mathbf{k}}(\mathbf{k}) \cdot d\mathbf{k}$$

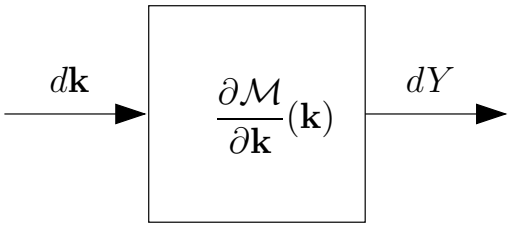


Figure B.2: Representation of the tangent model.

The adjoint model is defined as the adjoint operator of the tangent model. This can be represented as follows: $\left(\frac{\partial \mathcal{M}}{\partial \mathbf{k}}(\mathbf{k})\right)^* : \mathcal{Y}' \longrightarrow \mathcal{K}'$. It takes $dY^* \in \mathcal{Y}'$ an input variable and provides the adjoint variable $d\mathbf{k}^* \in \mathcal{K}'$ at output, see figure B.3:

$$d\mathbf{k}^* = \left(\frac{\partial \mathcal{M}}{\partial \mathbf{k}}(\mathbf{k})\right)^* \cdot dY^*$$

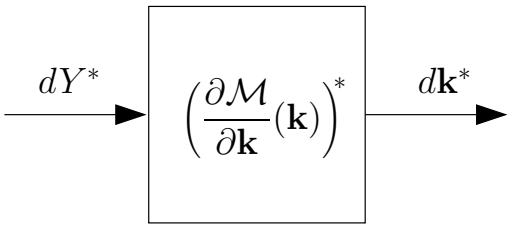


Figure B.3: Representation of the adjoint model.

Now, let us make the link between the adjoint code and the "computational gradient".

By definition of the adjoint, we have :

$$\left\langle \left(\tfrac{\partial \mathcal{M}}{\partial \mathbf{k}}\right)^* \cdot dY^*,\, d\mathbf{k} \right\rangle_{\mathcal{K}' \times \mathcal{K}} = \left\langle dY^*,\, \left(\tfrac{\partial \mathcal{M}}{\partial \mathbf{k}}\right) \cdot d\mathbf{k} \right\rangle_{\mathcal{Y}' \times \mathcal{Y}} \tag{B.1}$$

or, using the relations presented above:

$$\left\langle d\mathbf{k}^*,\, d\mathbf{k} \right\rangle_{\mathcal{K}' \times \mathcal{K}} = \left\langle dY^*,\, dY \right\rangle_{\mathcal{Y}' \times \mathcal{Y}} . \tag{B.2}$$

If we set $dY^* = (0,1)^T$ and by denoting the perturbation vector $d\mathbf{k} = (\delta y_0, \delta n, \delta z_b, \delta q^{in})^T$, we obtain:

$$\left\langle \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} dy^* \\ dJ^* \end{pmatrix} \right\rangle_{\mathcal{Y}' \times \mathcal{Y}} = \left\langle \begin{pmatrix} \delta y_0^* \\ \delta n^* \\ \delta z_b^* \\ \delta q^{in*} \\ \delta z_{out}^* \end{pmatrix}, \begin{pmatrix} \delta y_0 \\ \delta n \\ \delta z_b \\ \delta q^{in} \\ \delta z_{out} \end{pmatrix} \right\rangle_{\mathcal{K}' \times \mathcal{K}}$$

Moreover, we have by definition:

$$dj \;=\; \frac{\partial J}{\partial y_0}(\mathbf{k})\cdot\delta y_0 + \frac{\partial J}{\partial n}(\mathbf{k})\cdot\delta n + \frac{\partial J}{\partial z_b}(\mathbf{k})\cdot\delta z_b + \frac{\partial J}{\partial q_{in}}(\mathbf{k})\cdot\delta q_{in} + \frac{\partial J}{\partial z_{out}}(\mathbf{k})\cdot\delta z_{out}$$

Therefore, the adjoint variable $d\mathbf{k}^*$ (output of the adjoint code with $dY^* = (0,1)^T$) corresponds to the partial derivatives of the cost function j:

$$\begin{array}{lll} \frac{\partial J}{\partial y_0}(\mathbf{k}) \;=\; y_0^* & \frac{\partial J}{\partial n}(\mathbf{k}) \;=\; n^* & \\ \frac{\partial J}{\partial z_b}(\mathbf{k}) \;=\; z_b^* & \frac{\partial J}{\partial q_{in}}(\mathbf{k}) \;=\; q_{in}^* & \frac{\partial J}{\partial z_{out}}(\mathbf{k}) \;=\; z_{out}^* \end{array}$$

In summary, in order to compute the "computational gradient" (partial derivatives of the cost function j using differentiation of the forward code), first, one runs the direct code then one runs the adjoint code with $dY^* = (0,1)^T$ as input.

B.3 Automatic differentiation in short

Automatic differentiation softwares Automatic Differentiation (AD) (also called *algorithmic differentiation*) aims at computing the derivative of an program output. one the possible approach is the source-to-source one; it means the AD software (kind of a "super-over-compiler"...) transforms the direct source code into a tangent linear source code and/or an adjoint source code.
Researches in this field are very active; nowadays few reliable AD tools are available, both for Fortran and C languages. Let us cite only OpenAD and Tapenade. There use is still quite technical and tricky. It requires a good knowledge of what is the derivative of the output of the model, next how it is translated in terms of a program instructions.
In Fortran and C languages, the only remaining limitation is the pointer management; nevertheless some easy rules allow to circumvent this limitation.

We present below the basic principles of a AD tool, and we will focus on the Tapenade. Tapenade [23, 22] is an automatic differentiation tool for Fortran and C programs. It is developed by the Tropics team at INRIA Sophia-Antipolis. Tapenade works using source code transformation : it builds the tangent and/or adjoint code automatically from the direct source code. We refer to its rich webpage and the FAQ section.

Concerning AD in general, we refer also to [20, 34].

Some basic rules to respect for the direct code. In order to be able to differentiate a Fortran source code using Tapenade, one must respect some basic rules of code structure. Typically, the code structure must be as follows:

. Preprocessing.
Reading mesh, parameters, data etc AND allocate all arrays.

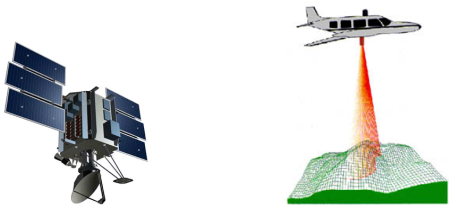
. Computational kernel.
  At the second level of the program tree, one gives the head routine to be differentiated to Tapenade. From this head routine, neither allocate nor pointer may appear.

. Post-processing.
  Writing in files, visualization etc.

Towards Data Assimilation

Parameters
- Manning, DEM (time independ.)
- B.C. (time dependent) + I.C.

Model math & num
Computational Code
(DassFlow - Telemac2D)



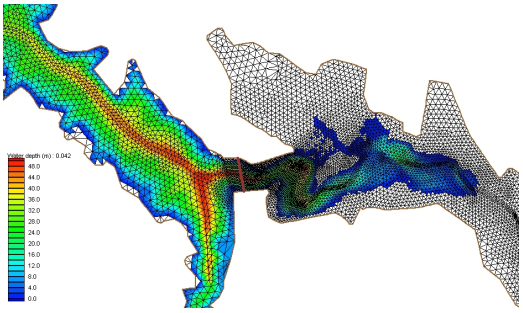
$$\frac{\partial(HU)}{\partial t} + \frac{\partial}{\partial x}\left(\beta_{xx} HUU + (\cos\alpha_x + \cos\alpha_z)^2 \frac{1}{2} gH^2\right)$$
$$-\Omega HV + \frac{1}{\rho}\left[\tau_{bx} - \tau_{sx} - \frac{\partial(H}{\partial}\right.$$
$$\frac{\partial(HV)}{\partial t} + \frac{\partial}{\partial y}\left(\beta_{xx} HVV + (\cos\alpha_y + \cos\alpha_z)^2 \frac{1}{2} gH^2\right)$$
$$-\Omega HU + \frac{1}{\rho}\left[\tau_{by} - \tau_{sy} - \frac{\partial(H}{\partial}\right.$$

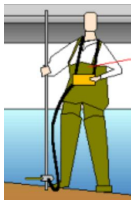
Observations:
measures in situ,
remotely-sensed

?

?

Simulated state
of the system
(h , q)





Variationnal D.A.:
Minimization discrepancy
(cost function)

Cost function:
It measures the discrepancy between the simulated state and the observations

It is a function of the control variable k (Manning, topo., B.C., I.C.):

$$j(k) = \frac{1}{2}\int_0^T \|C\, y(k;t) - y^{obs}(t)\|_*^2\, dt \;+\; \text{regularization terms}$$

where C: operator of observations

$\|.\|_*$: norm in the observation space.

Optimization problem:

$$\begin{cases} \text{Find } k^\star \text{ such that} \\ j(k^\star) = \min_K j(k) \end{cases}$$

where $j(k) = J(k, y^k)$ and y^k is the solution of the model (state of the system)

The state equation (forward model)

The control variable: $k = (y_0, v(t))^T$

y_0: I.C.; $v(t)$: parameters

State equation (nonlinear): Given k, find $y(t)$ s.t.

$$\begin{cases} \partial_t y(t) + A\big(v(t); y(t)\big) = 0 & \forall t \in]0,T[\\ y(0) = y_0 \end{cases}$$

This gives $y^k(t)$: the state, $j(k) = J(k; y^k)$: the cost function

Since the cost computation is time-consuming:
local minimization algorithm (quasi-Newton e.g. BFGS)
Since many variables of control:
introduction of the adjoint model to compute the gradient

Ref. JL Lions '68

Adjoint State equation Find $\tilde{y}(t)$ s.t.

$$\begin{cases} \partial_t \tilde{y}(t) - \left[\frac{\partial A}{\partial y}(y(t), v(t))\right]^* \tilde{y}(t) = C^* \Lambda_o \big(C\, y(t) - y^{obs}(t)\big) & \forall t \in]0,T[\\ \tilde{y}(T) = 0 \end{cases}$$

Note. a. Reverse time b.Contains the observations

Gradient obtained:

$$\frac{\partial j}{\partial y_0}(\mathbf{k}) = -\tilde{y}(0) + N(y_0 - y_b)$$

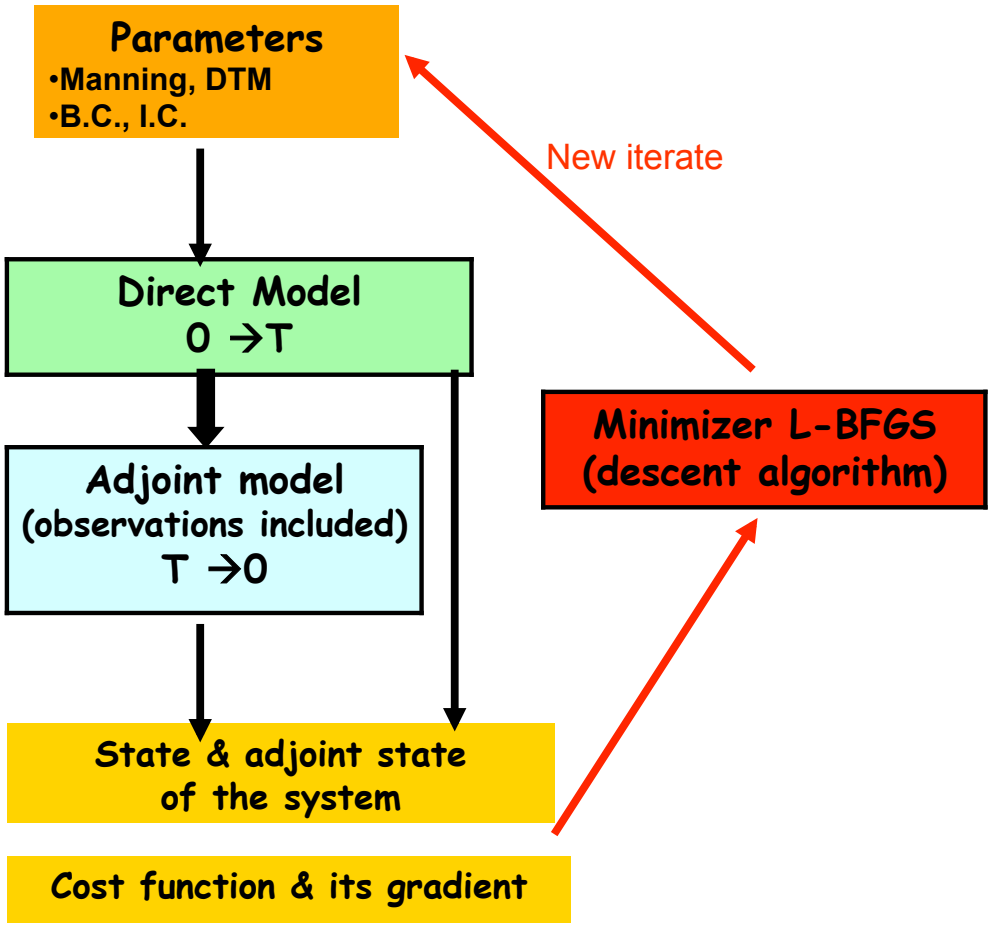
$$\frac{\partial j}{\partial v}(\mathbf{k}) = \left[\frac{\partial A}{\partial v}\Big(y(\mathbf{k}), v\Big)\right]^* \tilde{y}(\mathbf{k})$$

The Optimality System is

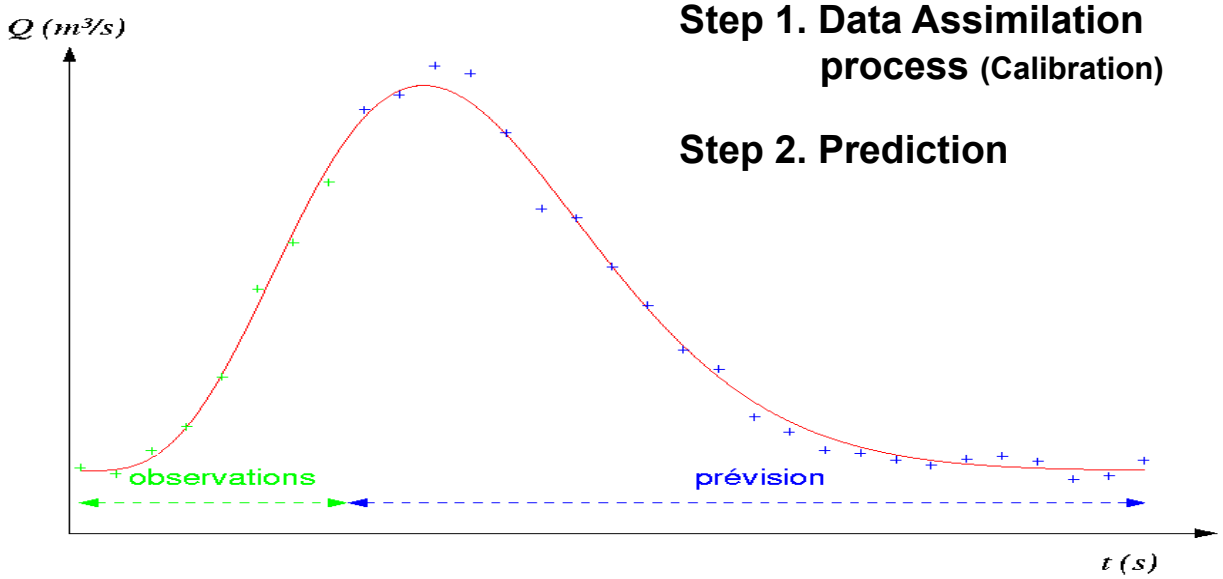
$$\begin{cases} \text{State equation} \\ \text{Ajoint State equation (contains observations)} \\ \nabla j(k) = 0 \end{cases}$$

Ref. e.g.
[Le Dimet – Talagrand '86]
[Talagrand-Courtier'87]

Variationnal Data Assimilation (4D-var): optimal control process



Data Assimilation Principle



Step 1. Data Assimilation process (Calibration)

Step 2. Prediction

Remarks

- The gradient gives local information.
  One run of the forward+adjoint models (without optimization) gives a **local sensitivity information**
- One can control the boundary conditions similarly

What Adjoint Code?
Differentiate / Discretize / Implement : in what order !?

- Continuous gradient:
  1. Differenciate mathematically; 2. discretize; 3. implement

- Discrete gradient:
  1. discretize; 2. differenciate; 3. implement

Make figure

- Code differenciation:
  1. Implement; 2. Differentiate the code (Automatic Differentiation tool e.g. Tapenade)

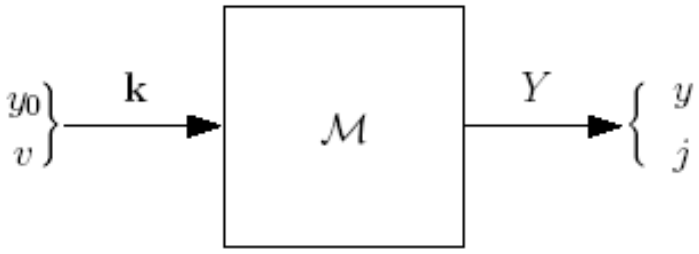
If the forward code well suited, prefer Automatic Differentiation (more re-usable).
NB. A.D. produces the gradient of the computed cost function

From Equations to Automatic Differentiation
Tapenade: source-to-source transformation

Ref.
M. Honnorat
PhD ' 07

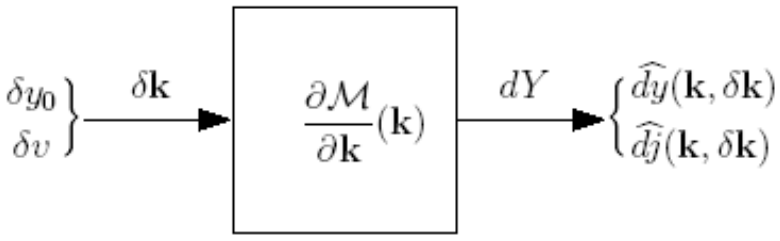
Forward model
(contains the cost function)

$$Y = \mathcal{M}(\mathbf{k}) \ .$$



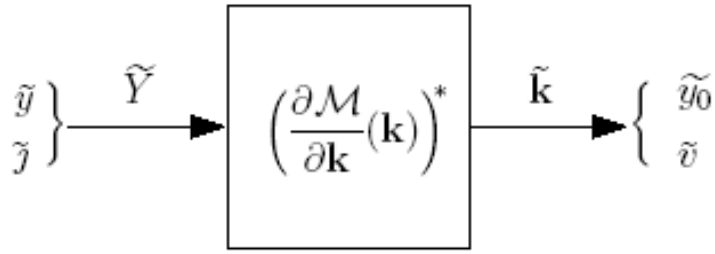
Linearized tangent model

$$dY = \left(\frac{\partial \mathcal{M}}{\partial \mathbf{k}}(\mathbf{k})\right) \cdot \delta\mathbf{k}$$



Adjoint model

$$\widetilde{\mathbf{k}} = \left(\frac{\partial \mathcal{M}}{\partial \mathbf{k}}\right)^* \cdot \widetilde{Y}$$



From Equations to Automatic Differentiation (continued)

Definition of the adjoint model: For all $\delta k \in \mathcal{K}, \tilde{Y} \in \mathcal{Y}$,

$$\left\langle \tilde{Y}, \left(\frac{\partial \mathcal{M}}{\partial \mathbf{k}}\right) \cdot \delta \mathbf{k} \right\rangle_{\mathcal{Y}} = \left\langle \left(\frac{\partial \mathcal{M}}{\partial \mathbf{k}}\right)^{*} \cdot \tilde{Y}, \delta \mathbf{k} \right\rangle_{\mathcal{K}}$$

Thus,

$$\langle \tilde{Y}, dY \rangle_{\mathcal{Y}} = \langle \tilde{\mathbf{k}}, \delta \mathbf{k} \rangle_{\mathcal{K}}$$

If we set $\tilde{Y} = (0,1)$ we get $\left\langle \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} \hat{dy} \\ \hat{dj} \end{pmatrix} \right\rangle_{\mathcal{Y}} = \left\langle \begin{pmatrix} \tilde{y_0} \\ \tilde{v} \end{pmatrix}, \begin{pmatrix} \delta y_0 \\ \delta v \end{pmatrix} \right\rangle_{\mathcal{X}}$

This means:

$$\hat{dj}(\mathbf{k}, \delta \mathbf{k}) = \frac{\partial j}{\partial y_0}(\mathbf{k}) \cdot \delta y_0 + \frac{\partial j}{\partial v}(\mathbf{k}) \cdot \delta v = \langle \tilde{y_0}, \delta y_0 \rangle_{H} + \langle \tilde{v}, \delta v \rangle_{\mathcal{U}}$$

Therefore the response of the adjoint code is the gradient

$$\frac{\partial j}{\partial y_0}(\mathbf{k}) = \tilde{y_0} \qquad \frac{\partial j}{\partial v}(\mathbf{k}) = \tilde{v}$$

Principles of « Automatic Differentiation »
Tapenade: source-to-source transformation

The « A.D. » (Algorithm Differentiation in fact) consists to compute derivatives of a given function M.
The derivatives are exact modulo rounding errors.

Ref. Hascoet et al. Tapenade tool INRIA Tropics

Given the control variable $\mathbf{k}_{\#} \subset X_0$, we have the forward code :

$$Y_{\#} = \mathcal{M}_{\#}(\mathbf{k}_{\#}) = m_p(X_{p-1}) \circ m_{p-1}(X_{p-2}) \circ \cdots \circ m_1(X_0)$$

Then in a direction $\delta \mathbf{k}_{\#}$

$$\frac{\partial \mathcal{M}_{\#}}{\partial \mathbf{k}_{\#}}(\mathbf{k}_{\#}) \cdot \delta \mathbf{k}_{\#} = m'_p(X_{p-1}) \times m'_{p-1}(X_{p-2}) \times \cdots \times m'_1(X_0) \cdot \delta \mathbf{k}_{\#}$$

where m'_p is the jacobian of the elementary operation m_p.

The tangent code value is computed from right to left (matrix-vector products).

This can be inserted into the original source code.

This is the « forward mode » or « tangent mode » of the A.D. tool

Principles of A.D. (continued)

Given the adjoint input variable $\widetilde{Y}_{\#}$, we have the adjoint code :

$$\left(\frac{\partial \mathcal{M}_{\#}}{\partial \mathbf{k}_{\#}}(\mathbf{k}_{\#})\right)^T \times \widetilde{Y}_{\#} = {m_1'}^T(X_0) \times {m_2'}^T(X_1) \times \cdots \times {m_p'}^T(X_{p-1}) \cdot \widetilde{Y}_{\#}$$

This can be computed from right to left (matrix-vector products).
Nevertheless,
• the code is reverse way
• all the variables have to be either stored or all recomputed
→ A balance between storing & recomputing is done (« checkpointing »)

This cannot be inserted
into the original source code !

This is the « adjoint mode »
or « reverse mode »of the A.D. tool

Source-to-source transformation: an extra source code is (automatically) generated

Tapenade On-line Differentiation engine.
Current version: all Fortran 95 excepted pointers & dynamic allocates

An example. The « tangent » mode

Ref. Hascoet et al.
Tapenade tool
INRIA Tropics

Instruction élémentaire du code initial (code direct): $a_i = x\, b_j + \cos(a_i).$

Ce qui peut s'écrire sous la forme:

$$m_k : \mathbb{R}^3 \longrightarrow \mathbb{R}^3$$

$$X_{k-1} = \begin{pmatrix} \texttt{a(i)} \\ \texttt{b(j)} \\ \texttt{x} \end{pmatrix} \longmapsto X_k = \begin{pmatrix} \texttt{a(i)} \\ \texttt{b(j)} \\ \texttt{x} \end{pmatrix}$$

On pose: $\delta X_k = m_k'(X_{k-1}) \cdot \delta X_{k-1}$ $\qquad m_k'(X_{k-1}) = \begin{pmatrix} \texttt{-SIN(a(i))} & \texttt{x} & \texttt{b(j)} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

alors:

$$\begin{pmatrix} \texttt{ad(i)} \\ \texttt{bd(j)} \\ \texttt{xd} \end{pmatrix} = \begin{pmatrix} \texttt{-SIN(a(i))} & \texttt{x} & \texttt{b(j)} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} \texttt{ad(i)} \\ \texttt{bd(j)} \\ \texttt{xd} \end{pmatrix}$$

code tangent : `ad(i) = -ad(i)*SIN(a(i)) + x*bd(j) + xd*b(j)`

Tangent code:
•Additional derivative instructions are inserted just before each instruction of the original code.
This leads to a source code quite easy to read.

•Differentiated variables retain the type and dimension of the original variable
•The code can be used either to compute the derivative in ONE direction given,
or to compute for ex. the jacobian of the flux (Newton algorithm)

An example. The « adjoint » mode

$$a_i = x\, b_j + \cos(a_i).$$

On a: $\widetilde{X}_{k-1} = m_k'^{T}(X_k) \cdot \widetilde{X}_k$

D'où

$$\begin{pmatrix} ab(i) \\ bb(j) \\ xb \end{pmatrix} = \begin{pmatrix} -SIN(a(i)) & 0 & 0 \\ x & 1 & 0 \\ b(j) & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} ab(i) \\ bb(j) \\ xb \end{pmatrix}$$

code adjoint :

```
xb    = xb + b(j)*ab(i)
bb(j) = bb(j) + x*ab(i)
ab(i) = -SIN(a(i))*ab(i)
```

Instructions are computed in reverse order.
Adjoint code: much more complicated to generate and
the source code is extremely difficult to read.
CPU time ~2-3 times higher

Principles of D.A. in C++: operator overloading. Tool of ref.: Adol-C.
Nevertheless, very hard in « adjoint» mode…

If and For statements

This includes functions like min, |.| etc

Direct mode

A; if (bool) then B else C; D;
1st branch: A; B; D . This gives: A'; A; B'; B; D'; D
2nd branch: A; C; D. This gives: A'; A; C'; C; D'; D
Hence:
A'; A; if (bool) then {B'; B} else {C' ; C} end if; D'; D;

Similarly A; for i=1:N, do B(i); D;

This gives: A'; A; for i=1:N, do {B'(i); B(i)}; D'; D;

Differentiable or not?
If the var. bool and/or the bounds of the for statement depend on the control var: not differentiable !
In practice, programs are piecewise differentiable; and non-differentiable points (rare !?)
should be identified by users …

In adjoint mode:
- branches of IF statements are treated separately,
  then the results are assembled after the derivation.
- we have to follow the graph of the program in reverse order
  => mind to have an equilibrated original call graph !
- all « intermediate solutions » (for all time step) are a-priori stored
  => very large memory required

=> Snapshot and checkpointing strategy

Snapshot and checkpointing in Tapenade

Store All (SA) strategy. Structure of the adjoint code:

1. **A forward sweep**, identical to the original program augmented with instructions that memorize the intermediate values before they are overwritten, and instructions that memorize the control flow in order to reproduce the control flow in reverse during the **backward sweep**.

2. **Start the backward sweep**, that actually computes the derivatives, and uses the memorized values both to compute the derivatives of each instruction and to choose its flow of control.

Active var: v1, v2, v3, v4

Original code

```
...
v2 = 2*sin(v1) + 5
if (v2>0.0) then
                v4 = v2 + p1*v3/v2
                p1 = p1 + 0.5
endif
call sub1(v2,v4,p1) ...
```

Forward Sweep

```
...
push(v2)
v2 = 2*sin(v1) + 5
if (v2>0.0) then
   push(v4) ; v4 = v2 + p1*v3/v2
   push(p1) ; p1 = p1 + 0.5
   push(true)
else push(false)
endif
push(v2) ; push(v4) ; push(p1)
call sub1(v2,v4,p1) ...
```

The isolated push are memorizations of intermediate values and control.

The group of 3 push before the subroutine call is a **snapshot** because the call will be **checkpointed**.

Checkpointing and snapshot in Tapenade

Differentiated var retain the type and dimension of the original variables

Forward Sweep

```
...
push(v2)
v2 = 2*sin(v1) + 5
if (v2>0.0) then
   push(v4) ; v4 = v2 + p1*v3/v2
   push(p1) ; p1 = p1 + 0.5
   push(true)
else push(false)
endif
push(v2) ; push(v4) ; push(p1)
call sub1(v2,v4,p1) ...
```

Backward Sweep

```
...
pop(p1) ; pop(v4) ; pop(v2)
call sub1_b(v2,v2b,v4,v4b,p1)
pop(control)
if (control) then
   pop(p1) pop(v4)
   v3b = v3b + v4b*p1/v2
   v2b = v2b + v4b*(1-p1*v3/(v2*v2))
   v4b = 0.0
endif
pop(v2)
v1b = v1b + v2b*2*cos(v1)
v2b = 0.0 ...
```

Remarks

- The adjoint of one original instruction is very often a sequence of instructions.
- sub1_b is the result of checkpointing on sub1, and therefore itself consists of a **forward sweep** followed by a **backward sweep**.
- Intrinsic functions are directly differentiated : no call to a sin_b procedure but to cos(v1).

B.4 Exercices

Let us consider a computational Fortran code solving a very simple dynamic system: the scalar linear ODE order one: $y(t) = f(y(t)u(t))$ with $f(y(t), u(t) = ay(t)+u(t)$, with initial condition. $y(t)$ is the state of the system, while $u(t)$ is the control.

The goal here is to differentiate first by hand the source code (algorithmic differentiation), next using Tapenade software (automatic differentiation).

The direct source code is the following.

```
! Basic program designed to be differentiated using Tapenade
! The head of the tree to be differentiated by Tapenade is
! the first routine of computational part
program dynamic_system
  implicit none
  integer i
  real*8 :: dt, T0, TN, y0
! all allocate must be in the main program
! (out of the computational part which will be diffrentiated)
  real*8, dimension(:), allocatable :: u, y
  integer :: nstep
  T0 = 0.d0
  TN = 1.d0
  !time step (should be defined by the stability condition)
  dt = 0.001d0
  nstep = int((TN-T0)/dt)
  !allocate instructions must be out of the part differentiated by Tapenade
  allocate(y(nstep))
  allocate(u(nstep))
  !initial state and initial control
  !here they are constant (u could be a time dependent vector)
  y0 = 3.4d0
  u  = 2.6d0

  !head routine of the computational part
  !=>the top tree routine of the code thar will be differentiated by Tapenade
  call solve_model( y0, u, y, dt, nstep)

end program dynamic_system

!The direct model is:
!y'(t) = f(y(t)u(t)) + I.C.
!Scheme: Euler explicit
subroutine solve_model( y0, u, y, dt, nstep )
```

```
  implicit none
  integer, intent(in) :: nstep
  real*8,  intent(in) :: dt, y0
  real*8,  dimension(nstep), intent(in)  :: u
  real*8,  dimension(nstep), intent(out) :: y
  real*8 :: f
  integer :: i
  ! initialize the state variable
  y(1) = y0
  ! time loop
  do i = 1, nstep - 1
     y(i+1) = y(i) + dt * f(y(i),u(i))
  end do
  return
end subroutine solve_model

! Right-hand side f
! f(y(t),u(t)) = a y(t) + u(t)
real*8 function f(y,v)
  implicit none
  real*8, intent(in)  :: y, v
  real*8, parameter :: a = 1
  f =  a * y + v
end function f
```

Exercice B.1. *Algorithmic differentiation by hand.*
1) Write the tangent linear codes of the subroutine solve_model and function f.
2) Write the corresponding adjoint codes.

Exercice B.2. *Algorithmic differentiation using Tapenade.*
1) Add in the direct code the computation of a cost function j.
2) Use Tapenade software (web interface on-line) in order to obtain:

a) *the tangent linear codes of the subroutine solve_model and function f.*

b) *the corresponding adjoint codes.*

3) Read carefully and analyze the two routines automatically generated.

B.5 Recycle your linear solver

Let us recall that, roughly, the adjoint of a linear system is the same linear system but transposed. (Notice that the latter feature brings some quite heavy difficulties...). Thus, one should not derive the linear solver instructions one-by-one using an automatic differentiation tool (source-to-source) but instead recycle it ! Here, we describe how to generate the adjoint of a

routine containing a call to a linear solver.

Next, we present a small Fortran program including the call to a linear solver; we apply the method on the codes automatically generated by Tapenade.

B.5.1 How to adjoint-ize your linear solver ?

The direct routine

Let us consider two input parameters c and d (hence *active variables* in Tapenade terminology). Then, we have A matrix and b vector defined as follows:

$$\begin{pmatrix} A \\ b \end{pmatrix} = f(c,d) = \begin{pmatrix} f_1(c,d) \\ f_2(c,d) \end{pmatrix}$$

Let x be the vector solution of the linear system: $Ax = b$. Next, we compute the cost function j.

We present in Fig. B.4 the direct routines dependencies.

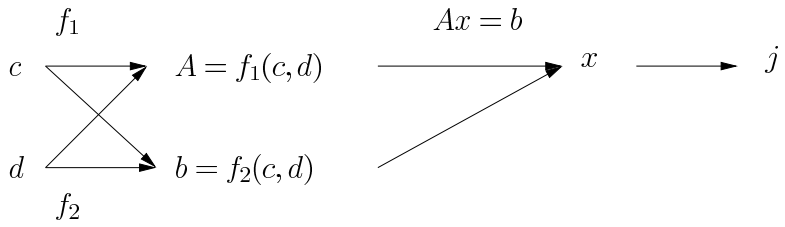


Figure B.4: Direct routines representation

The linear tangent routine generated

The linear tangent code derived by Tapenade is defined as follows:

- $\dot{c}$ and $\dot{d}$ are the corresponding tangent variables of the parameters, they are input parameters.
- $\dot{A}$ and $\dot{b}$ satisfy:

$$\begin{pmatrix} \dot{A} \\ \dot{b} \end{pmatrix} = df(c,d) \cdot \begin{pmatrix} \dot{c} \\ \dot{d} \end{pmatrix}$$

Next, we have to treat the linear solver. If we differentiate the linear system:

$$Ax = b \tag{B.3}$$

we find: $A\dot{x} + \dot{A}x = \dot{b}$ or, after rewrite:

$$A\dot{x} = \dot{b} - \dot{A}x \tag{B.4}$$

At this stage, we already know A and x by running direct routine. The variables $\dot{A}$, $\dot{b}$ are output variables of the Tapenade tangent linear routines. Next, we call the linear solver in order to obtain: $\dot{x}$. The latter represents the derivative value of x at point (c, d) in the direction $(\dot{c}, \dot{d})$ given.
Thus, we are able to compute $\dot{x}$ without deriving the linear solver instructions. We can call directly the same linear solver (we recycle it).
Next, we obtain $\dot{j}$ the gradient value at point (c, d) in the direction $(\dot{c}, \dot{d})$ given (it is a scalar value like j).
We present in Fig. B.5 the scheme representing the linear tangent routines.

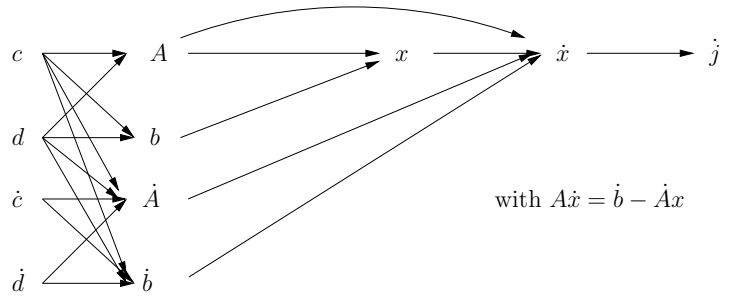


Figure B.5: Linear tangent routines representation

The adjoint routines generated

Let us recall that the adjoint code is deduced from the tangent linear code, in the reverse order. The output variable of the adjoint routine are $\bar{c}$ and $\bar{d}$. In a Tapenade point of view, they are the adjoint variables of (c, d) (same type); while in a mathematical point of view, they are the *gradient* with respect to c and d.
The input variable of the adjoint cost computation routine is $\bar{j}$. Let us recall that $\bar{j}$ must be set to 1, see Section B.2.
The input variable of the adjoint linear system routine is $\bar{x}$.

The adjoint code can be splitted in three steps, see Fig. B.5:

1) From $\bar{j}$ to $\bar{x}$ (let us assume it is a separated routine: cost computation, adjoint)

2) From $\bar{x}$, find $\bar{A}$ and $\bar{b}$

3) From $\bar{A}$ and $\bar{b}$, find $\bar{c}$ and $\bar{d}$.

The first and third steps can be obtained directly by running the Tapenade adjoint code. The adjoint of the third step writes:

$$\begin{pmatrix} \bar{c} \\ \bar{d} \end{pmatrix} = df^*(c, d) \cdot \begin{pmatrix} \bar{A} \\ \bar{b} \end{pmatrix} \tag{B.5}$$

The adjoint of the linear system

The second step involves a call to the linear solver. Thus, let us detail how to compute $\bar{A}$ and $\bar{b}$ with the use of a linear solver as a black box. In other words, we detail below the adjoint of Step 2) only.
Input variable is $\bar{x}$, while output variables are $\bar{A}$, $\bar{b}$.
In the linear tangent code, we have : $A\dot{x} = \dot{b} - \dot{A}x$ or equivalently:

$$\begin{array}{lrcl} 2a) & \dot{b'} & = & \dot{b} - \dot{A}x \\ 2b) & A\dot{x} & = & \dot{b'} \end{array}$$

The adjoint is reverse. Thus let us consider first the instruction 2b) $A\dot{x} = \dot{b'}$ only. It can be written as follows (see e.g. [22]):

$$\begin{pmatrix} \dot{x} \\ \dot{b'} \end{pmatrix} = \begin{pmatrix} 0 & A^{-1} \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} \dot{x} \\ \dot{b'} \end{pmatrix} \tag{B.6}$$

Two fundamental remarks.
a) Let us point out that by convention, at left-hand sides are the variables at output, while at right-hand sides are the variables at input (of the routine).
b) Let us point out that input variables in the linear tangent routine become output variables in the adjoint routine. Furthermore, by convention, output variables are initialized at 0.

Let us write the adjoint of the instruction 2b). Since $\dot{b'}$ is the input variable of the instruction, its adjoint $\bar{b'}$ will be the output one (and set at 0 when entering into the routine). Similarly, since $\dot{x}$ is the output variable, its adjoint $\bar{x}$ will be the input one. The adjoint instruction of (B.6) writes:

$$\begin{pmatrix} \bar{x} \\ \bar{b'} \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ A^{-T} & 1 \end{pmatrix} \times \begin{pmatrix} \bar{x} \\ \bar{b'} \end{pmatrix}$$

Hence:

$$\left\{ \begin{array}{lcl} \bar{x} & = & 0 \\ \bar{b'} & = & A^{-T}\bar{x} + \bar{b'} \end{array} \right.$$

Thus the system rewrites:

$$\left\{ \begin{array}{lcl} A^T\bar{b'} & = & \bar{x} \\ \bar{x} & = & 0 \end{array} \right.$$

And one has to solve the linear system: $A^T\bar{b'} = \bar{x}$. It can be done using the linear solver called as a black box (we recycle it !). It gives $\bar{b'}$.

Now, let us consider the adjoint of the instruction 2a):

$$2a) \quad \dot{b'} = \dot{b} - \dot{A}x \tag{B.7}$$

This linear instruction writes as follows:

$$(\dot{b}', \dot{b}, \dot{A}) = (\dot{b}', \dot{b}, \dot{A}) \times \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ -x & 0 & 1 \end{pmatrix}$$

The corresponding adjoint instruction writes:

$$(\bar{b}', \bar{b}, \bar{A}) = (\bar{b}', \bar{b}, \bar{A}) \times \begin{pmatrix} 0 & 1 & -x^T \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Therefore, the adjoint instruction of (B.7) writes:

$$\left\{ \begin{array}{rcl} \bar{b}' & = & 0 \\ \bar{b} & = & \bar{b}' + \bar{b} \\ \bar{A} & = & -\bar{b}' x^T + \bar{A} \end{array} \right.$$

The variables $\bar{A},\ \bar{b}$ are output variables, hence set at 0 when entering into the adjoint routine.

Therefore, in summary the adjoint of the tangent linear instructions $A\dot{x} = \dot{b} - \dot{A}x$ (ie Step 2)) can be re-write:

$$\left\{ \begin{array}{rcl} A^T \bar{b} & = & \bar{x} \\ \bar{A} & = & -\bar{b} x^T \\ \bar{x} & = & 0 \end{array} \right.$$

The first instruction can be solved using the same linear solver than the direct routine ("recycle your linear solver"). The second instruction writes too: $\bar{A}_{ij} = -\bar{b}_i x_j$.

Remark B.3.
- Let us notice that the matrix $\bar{A}$ is the same type of A with the same sparse profile (even if $-\bar{b}x^T$ is a-priori a full matrix). As a matter of fact, one needs only the adjoint values of coefficients $A_{i,j}$ (and one do not need the others coefficients).

- Let us recall that $(\bar{c}, \bar{d})$ are the components of the gradient with respect to (c, d) respectively; they are obtained by (B.5).

- Since the direct model is $Ax = b$, the state of the system is x, the adjoint state is $\bar{b}$ (the solution of the transposed linear system) and $\bar{x}$ is the right-hand side which contains the observations and misfit terms (it equals to $(x - x^{obs})$ if the observation operator equals identity).

We summarize all steps in Fig. B.6.

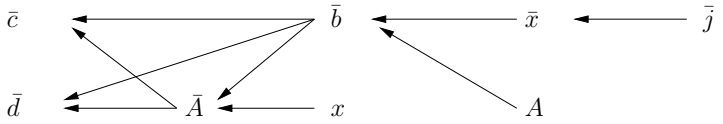


Figure B.6: Adjoint routine representation

B.5.2 A simple example

We illustrate the previous procedure of the linear solver recycling on a very simple example. In all the following, one can replace the routine *sys_solver* by any linear system solver routine. This example has been written by R. Madec, research engineer, IMT Toulouse.

Direct code

We present the Fortran source code of the direct subroutine written in file *simple_operation.f90*:

```
subroutine simple_operation(c,d,x)
  implicit none
  integer, parameter :: n=2
  double precision, intent(in) :: c
  double precision, intent(in) :: d
  double precision, dimension (n), intent(out):: x
  double precision, dimension (n,n) :: A
  double precision, dimension (n)   :: b

!definition of matrix A
  A(1,1) = 3*c +4*d
  A(1,2) = c +18*d
  A(2,1) = 2.5*d
  A(2,2) = 3.4*c

!definition of vector b
  b(1) = 8*c*d
  b(2) = 2*d

!solver Ax=b
  call sys_solver(A,b,x,n)

end subroutine simple_operation
```

Linear tangent code

We generate the linear tangent code using Tapenade. To do so, we type in a terminal (or in a Makefile):

```
tapenade -forward -head simple_operation \
    -vars "c d" -outvars "x" \
```

```
    -html -O $(PWD)/forward simple_operation.f90
```

Then, the file *simple_operation_d.f90* is created in the *forward* directory. This file is the following:

```
!        Generated by TAPENADE     (INRIA, Tropics team)
!  Tapenade 3.4 (r3375) - 10 Feb 2010 15:08
!
!  Differentiation of simple_operation in forward (tangent) mode:
!   variations   of useful results: x
!   with respect to varying inputs: c d
!   RW status of diff variables: x:out c:in d:in
SUBROUTINE SIMPLE_OPERATION_D(c, cd, d, dd, x, xd)
  IMPLICIT NONE
  INTEGER, PARAMETER :: n=2
  DOUBLE PRECISION, INTENT(IN) :: c
  DOUBLE PRECISION, INTENT(IN) :: cd
  DOUBLE PRECISION, INTENT(IN) :: d
  DOUBLE PRECISION, INTENT(IN) :: dd
  DOUBLE PRECISION, DIMENSION(n), INTENT(OUT) :: x
  DOUBLE PRECISION, DIMENSION(n), INTENT(OUT) :: xd
  DOUBLE PRECISION, DIMENSION(n) :: b
  DOUBLE PRECISION, DIMENSION(n) :: bd
  DOUBLE PRECISION, DIMENSION(n, n) :: a
  DOUBLE PRECISION, DIMENSION(n, n) :: ad
  INTEGER :: i,j

  ad(1, 1) = 3*cd + 4*dd
  a(1, 1) = 3*c + 4*d
  ad(1, 2) = cd + 18*dd
  a(1, 2) = c + 18*d
  ad(2, 1) = 2.5*dd
  a(2, 1) = 2.5*d
  ad(2, 2) = 3.4*cd
  a(2, 2) = 3.4*c
  bd(1) = 8*(cd*d+c*dd)
  b(1) = 8*c*d
  bd(2) = 2*dd
  b(2) = 2*d

  CALL SYS_SOLVER_D(a, ad, b, bd, x, xd, n)

END SUBROUTINE SIMPLE_OPERATION_D
```

Using the method presented in subsection B.5.1, we replace the unknown linear tangent of the linear system solver:

```
  CALL SYS_SOLVER_D(a, ad, b, bd, x, xd, n)
```

by

```
  CALL SYS_SOLVER(a, b, x, n)
  !matrix vector product : bd = bd - Ad x
  do j=1,n
     do i=1,n
       bd(i)=bd(i) - ad(i,j)*x(j)
     end do
  end do
  CALL SYS_SOLVER(a, bd, xd, n)
```

Adjoint code

We generate the adjoint code using Tapenade. To do so, we type in a terminal (or in a Makefile):

```
tapenade -backward -head simple_operation \
    -vars "c d" -outvars "x" \
    -html -O $(PWD)/backward simple_operation.f90
```

Then, the file *simple_operation_b.f90* is created in the *backward* directory. This file is the following:

```
!        Generated by TAPENADE     (INRIA, Tropics team)
!  Tapenade 3.4 (r3375) - 10 Feb 2010 15:08
!
!  Differentiation of simple_operation in reverse (adjoint) mode:
!   gradient     of useful results: x
!   with respect to varying inputs: x c d
!   RW status of diff variables: x:in-zero c:out d:out

SUBROUTINE SIMPLE_OPERATION_B(c, cb, d, db, x, xb)
  IMPLICIT NONE
  INTEGER, PARAMETER :: n=2
  DOUBLE PRECISION, INTENT(IN) :: c
  DOUBLE PRECISION :: cb
  DOUBLE PRECISION, INTENT(IN) :: d
  DOUBLE PRECISION :: db
  DOUBLE PRECISION, DIMENSION(n) :: x
  DOUBLE PRECISION, DIMENSION(n) :: xb
  DOUBLE PRECISION, DIMENSION(n, n) :: a, at
  DOUBLE PRECISION, DIMENSION(n, n) :: ab
  DOUBLE PRECISION, DIMENSION(n) :: b
  DOUBLE PRECISION, DIMENSION(n) :: bb
  INTEGER :: i,j
  a(1, 1) = 3*c+ 4*d
```

```
  a(1, 2) = c + 18*d
  a(2, 1) = 2.5*d
  a(2, 2) = 3.4*c
  b(1) = 8*c*d
  b(2) = 2*d

  CALL SYS_SOLVER_B(a, ab, b, bb, x, xb, n)

  db = 2*bb(2)
  bb(2) = 0.D0
  cb = 3.4*ab(2, 2) + 8*d*bb(1)
  ab(2, 2) = 0.D0
  db = db + 2.5*ab(2, 1) + 8*c*bb(1)
  ab(2, 1) = 0.D0
  cb = cb + ab(1, 2)
  db = db + 18*ab(1, 2)
  ab(1, 2) = 0.D0
  cb = cb + 3*ab(1, 1)
  db = db + 4*ab(1, 1)
  xb = 0.D0
END SUBROUTINE SIMPLE_OPERATION_B
```

Using the method presented in subsection B.5.1 we replace the unknown adjoint of the linear system solver:

```
  CALL SYS_SOLVER_B(a, ab, b, bb, x, xb, n)
```

by the following instructions:

```
!at = transpose(a)
  at(1, 1) = 3*c + 4*d
  at(2, 1) = c + 18*d
  at(1, 2) = 2.5*d
  at(2, 2) = 3.4*c

!new solver
  call sys_solver(a,b,x,n)
  call sys_solver(at,xb,bb,n)
  ab=0.d0
  do j=1,n
     do i=1,n
 if (ab(i,j).ne.0.) ab(i,j)=-bb(i)*x(j)
     end do
  end do

!At = transpose(A)
  do j=1,n
```

```
   do i=1,n
    At(i, j) = A(j,i)
   end do
  end do
!new solver
  call linear_solver(A,b,x,n)
  call linear_solver(At,xb,bb,n)
  do j=1,n
     do i=1,n
  if (Ab(i,j).ne.0.) Ab(i,j)= - bb(i)*x(j)
     end do
  end do
```

We have recycled the linear solver (which can be called as a black box).

B.6 Complement: MPI instructions

We present the procedure to obtain the adjoint code of a Fortran code calling MPI basic routines and using the automatic differentiation tool Tapenade.
This part is skipped since no time enough.

B.7 Complement: optimize your memory. Checkpointing.

When you use Tapenade to create an adjoint code, you have no mean to know the memory consumption that will take Tapenade. The latter is a direct consequence of the ''PUSH" and "POP" instructions introduced by Tapenade in your code. The "PUSH" instruction is keeping in memory the variable until a "POP" of this variable is done. Every adjoint code is divided into two part: the first part, which is just a copy of your direct code with add of "PUSH" instructions filling the memory; the second part written by Tapenade and running a backward code with the adjoint variables, and freeing the memory with POP instructions.
Memory problems can appear if you have "PUSH"es of, for example, your state variable at every time steps on a simulation.

We present below a way to avoid memory faults by using the checkpointing strategy of Tapenade in a "smart" way.

Checkpointing strategy First thing is to anticipate and identify which part of the "PUSH" and "POP" instructions in your code is potentially greedy in memory. To narrowing it, it should appear in the biggest loop (i.e. with the most iterations) of the algorithm and occur where your state variables are.

An example of order of magnitude. In DassFlow software, based on the 2D shallow-water equations, the main "PUSH" and "POP" contributions are related to the state variable in the time step loop.
For a 25 000 cells mesh with 100 000 time steps, if we do nothing, as our state variable counts 3 fields, this would cost just for "pushing" this variable: $3 \times 25000 \times 100000 \times 8 = 6 \times 10^{10}$o= 60 Go of RAM which is a lot !

(NB. The number 8 represents the octet size of a double precision number; it is 4 for a simple precision).
Most of the time, it is only one variable that make your system fall.

B.8 Practical work Tapenade

As practical work, we generate the adjoint code of a Fortran source code using the software Tapenade. Next, validate the codes obtained and we perform sensitivities analysis.

The direct code solves the 1D heat equation $\partial_t u = K\partial_x^2 u$ using finite differences, with a time scheme either explicit in time or implicit in time. In the latter case, we call a linear solver (as a black box).

In the practical work, we consider two cases (two direct programs):

- an explicit FD solver with MPI instructions.
- an implicit FD solver with a call to a linear solver as a black box.

Sorry, no time enough to consider this part.

B.9 Example of a software architecture designed for Automatic Differentiation

See the toy DassFlow code (TP automatic differentiation using Tapenade software).

Appendix C

Weather forecasting & VDA algorithm variants *

The outline of this chapter is as follows.

Contents

C.1 Data for weather forecasting

The historical and one of the most challenging DA problem: weather forecast

The Variational Data Assimilation (VDA) method has been introduced by Y. Sasaki in the mid 50's [37, 38] for applications in meteorology. An improvement of the method based on the optimal control theory and the adjoint equations has been introduced by G. Marchuk, F.-X. Le Dimet and O. Talagrand in the mid 80's, see e.g. [27] and references therein; see also the

pioneer studies [15, 31, 30].

Since the early 2000th, operational VDA systems are implemented in all large Weather Prediction Centers (WPC, eg. ECMWF, Meteo-France, Met Office, NOAA etc). The most complex and sophisticated data assimilation systems are probably elaborated in these WPC. The dynamic of the atmosphere is 3D, turbulent, multi-scale, multi-physics. The mathematical models are based on the conservation laws: mass, momentum, energy. The resulting equations are highly non-linear, fully coupled. Unknowns are the following 3D fields: temperature, pressure, humidity, velocities etc. The complexity of the (direct) model is huge: at each time step (typically 10 minutes for mid-range forecast), the discrete unknown contains billions of components.

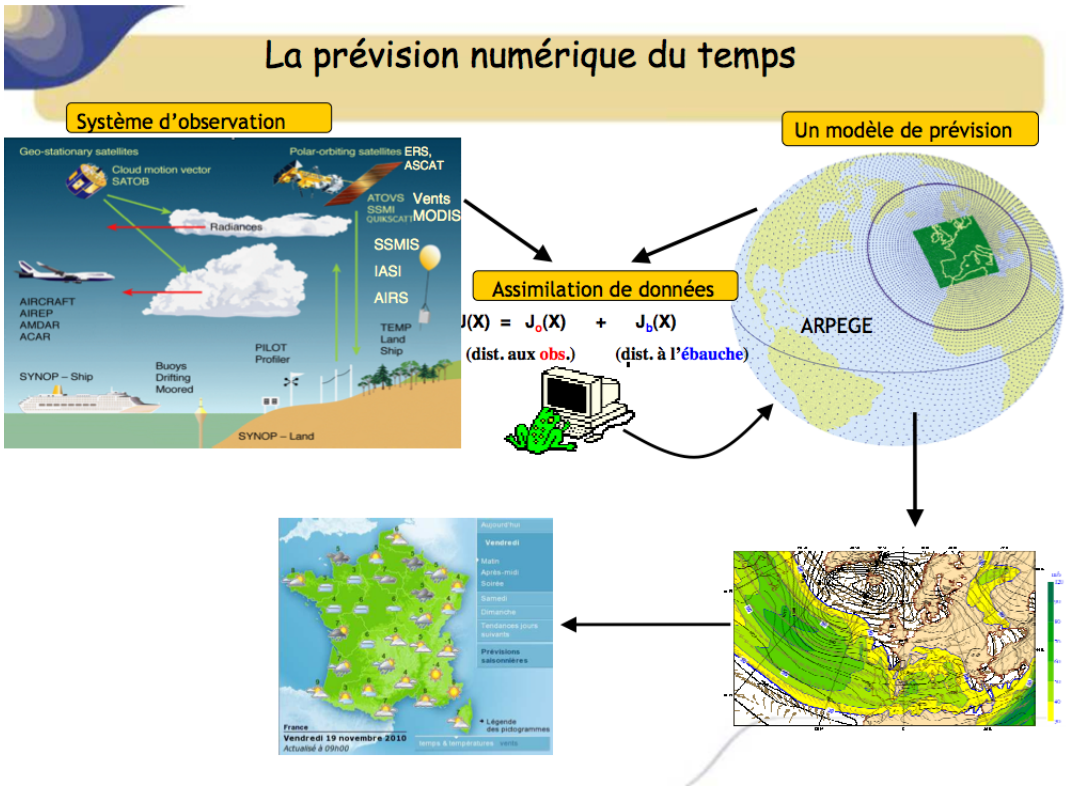


Figure C.1: Weather prediction based on data assimilation processes optimally fusing all the available information. Image source: CNRM, Meteo-France.

Forecasts have been greatly improved during the last two decades in particular due the introduction of more and more sophisticated data assimilation processes. The latter enable to benefit of the exponentially exploding observations systems. Employed algorithms are variants of the presented VDA algorithm combined with the Ensemble Kalman Filter (EnKF) (not studied in the present course). Some information on weather forecasting are presented in Appendix 5), .

Observations are heterogeneous in nature, resolution and accuracy; they are distributed more or less regularly in space and time. The numbers of available observations is exploding; it

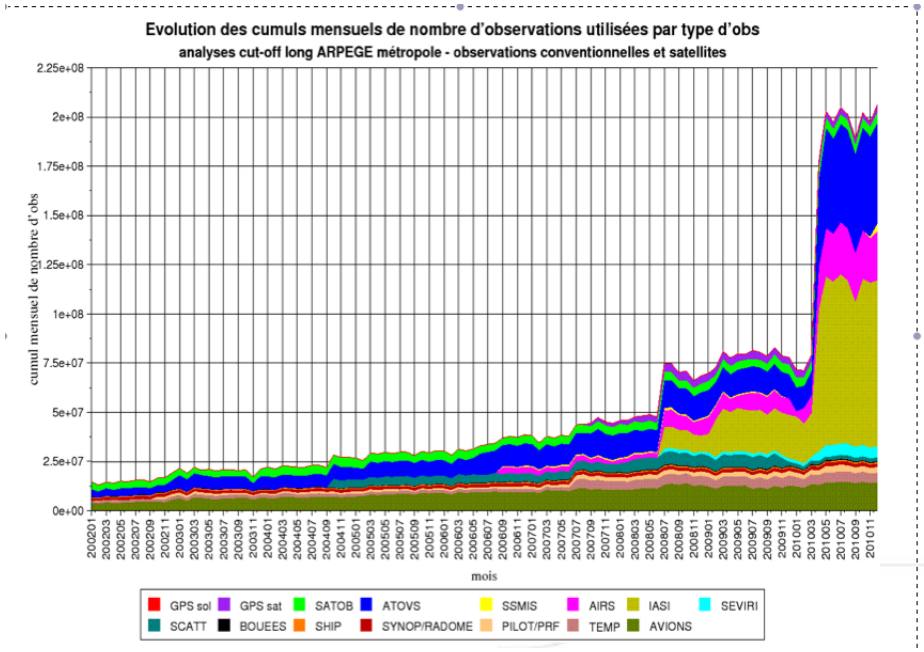


Figure C.2: Observations number explosion. Plot: numbers used in the global scale model ARPEGE (and classified by nature) vs months (period 2002-2010). Image source: CNRM, Meteo-France.

is about dozens of millions per day, see Fig. Satellite measurements (surface temperature, nebulosity, radiance etc) are difficult to compare to the model outputs. Measurements in-situ (temperature, pressure, humidity, wind) are heterogeneously distributed and very local values. In complex systems, both observations and models are affected by multi-source uncertainties. Furthermore, *extra difficulties are specific to geophysical flows: it is non reproducible phenomena, observations are sparse and uncertain, phenomena are multi-scale and models of course incomplete; moreover initial states cannot be fully measured.*

A great challenge of the WPC is to reconstruct the initial state of the atmosphere i.e. the atmosphere variables at present time ! Indeed even if the observations are more and more dense and accurate, they are not dense enough to accurately reconstruct the atmosphere state everywhere. Moreover the atmosphere dynamic depends strongly on its initial state; actually it is well known its behavior may be chaotic in medium-long time range.

The *purpose of data assimilation* is to reconstruct as accurate as possible the state of the atmosphere at present time (the initial condition) using all available information i.e.:

. the physical and conservation laws governing the flow dynamic (mathematical model),

. the observations (sparse and heterogeneous in nature),

. the statistic errors of the observations and prior probabilistic behavior of the various fields.

The *VDA algorithms aim at fusing as best as possible all these information in view to recover the atmosphere state at present time.*

Orders of magnitude from the operational ECMWF forecast system (in 2009...)

Below are presented some order of magnitudes for Numerical Weather Prediction (NWP).
All information and figures below come from documents written by Y. Tremolet and M. Fisher from the European Centre for Medium-Range Weather Forecasts (ECMWF) based at Reading, UK. ("ECMWF is an intergovernmental organization supported by 34 States. It provide operational medium- and extended-range forecasts and a state-of-the-art super-computing facility for scientific research. It pursues scientific and technical collaboration with satellite agencies and with the European Commission").

Forecast has improved the last twelve years in particular because of the introduction of data assimilation process into the complex dynamical atmosphere models, which allow to benefit more and more from the global various observing systems. Methods employed are 4d-var, 3d-var and ensemble methods.

Orders of magnitude presented below date from 2009 - 2010.

The unknowns of the (non-linear, fully coupled) mathematical models are 3D fields: temperature, pressure, humidity, velocities.
The dimension of the state (number of unknowns) is about 10^9.
Time step is about 10 minutes. Discretization is about 16 kms in horizontal (triangles) and with about 90 layers in vertical (0 - 80 km meshed).

Observations are heterogeneous in nature and in space. They are in dozens of millions (see figures below).
Satellites measurements are: surface temperature, nebulosity etc. They are complex to interpret and to compare to the model outputs.
Measurements in-situ are: temperature, pressure, humidity, wind etc.

The configuration uses a 12h cycling window, with a 4D-var incremental method.
The outer loop (and forecast) resolution is 25 km. The inner loops resolutions are between 200 km and 80 km.
On average, 9 million observations are assimilated per 12h cycle. 96% of assimilated data is from satellites. On average, 4D-Var runs on 1536 CPUs in 1h10.
The incremental method with appropriate preconditioning allows the computational cost to be reduced to an acceptable level.

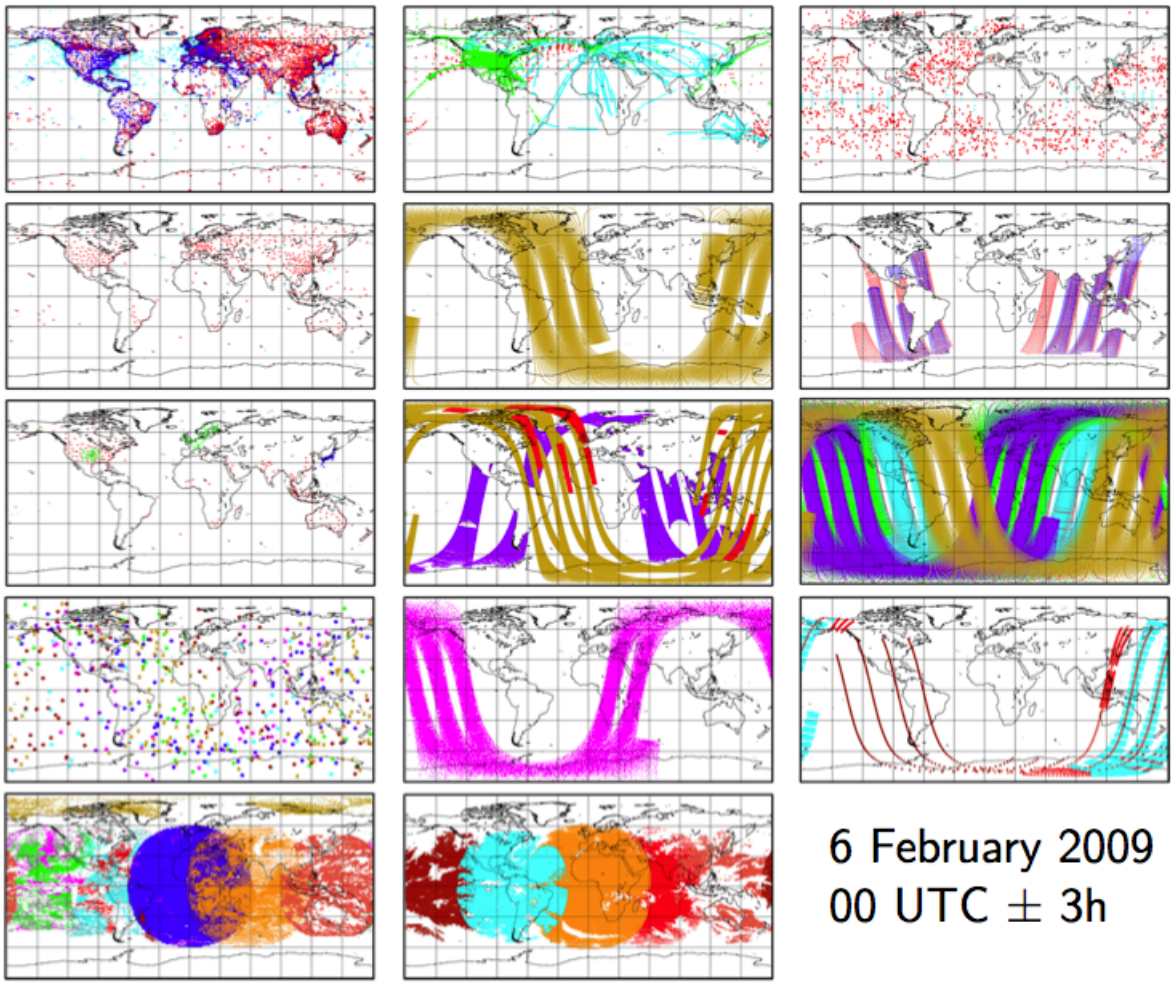


Figure C.3: Meteorology. Some observation coverage. Figure extracted from Tremolet - Fisher, ECMWF, Colloquium 2009.

C.2 Discrete formalism of the VDA equations

In next section, classical variants of the original VDA algorithm (classically called "4D-Var" algorithm) presented in previous chapters are derived, in particular the so-called "incremental 4D-var" algorithm, the 3D-FGAT algorithm and others.
To do so, first the VDA equations are derived in a discrete formalism hence not requiring differential calculs skills. Thus, this section requires a lower mathematical background compared to the previous ones.

Let us rewrite the variational data assimilation problem and the resulting optimality system in its semi-discrete and classical form (on the contrary to the continuous weak forms). For such a presentation, the reader can refer to the ECMWF data assimilation course available on-line (www.ecmwf.int) or [43].

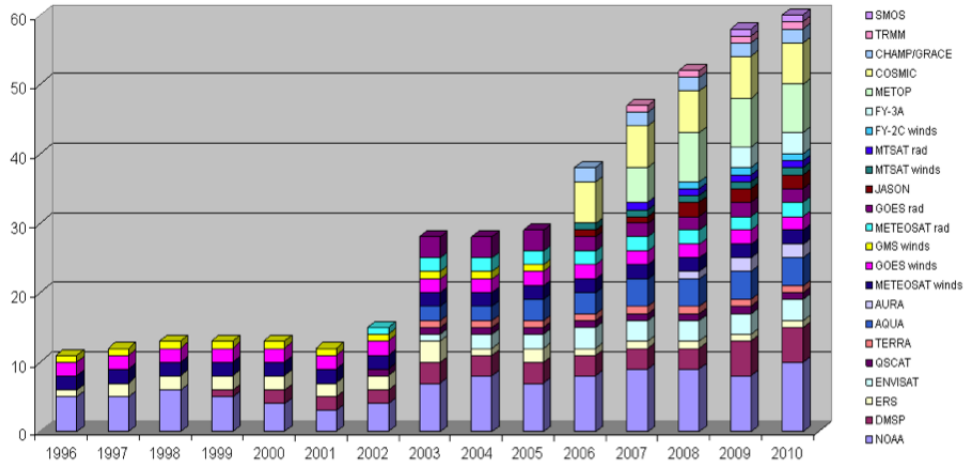


Figure C.4: Meteorology. Observation sources. Figure extracted from Tremolet - Fisher, ECMWF, Colloquium 2009.

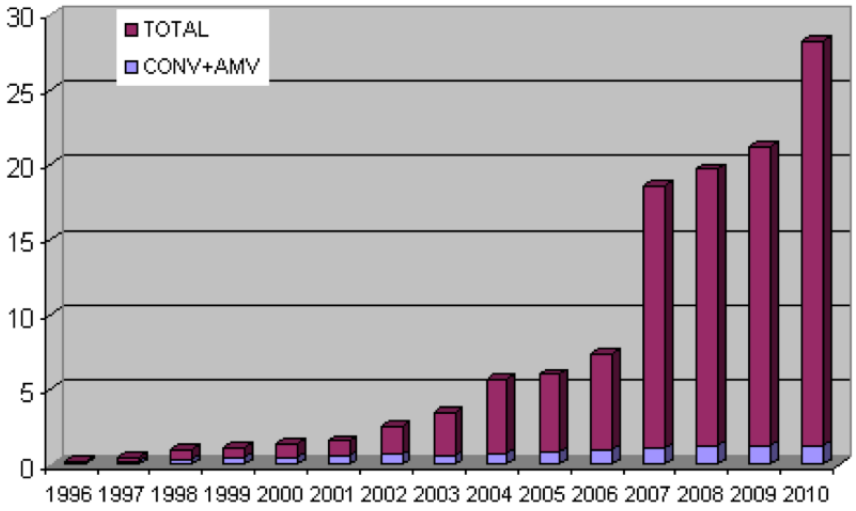


Figure C.5: Meteorology. Observation Numbers (in milions per day). Source: ECMWF.

In the present section, the direct model reads as follows :

$$(D_d)\begin{cases} \text{Given the initial condition } y_0 \in \mathbb{R}^n, \text{ find } \{y_n\}_n \in \mathbb{R}^n \text{ such that :} \\ y_{n+1} = \mathcal{M}_n(y_n) \quad n = 0, 1, ..., N \text{ (time steps)} \\ y_0 \text{ given} \end{cases} \tag{C.1}$$

where $\mathcal{M}_n(y_n)$ denotes the non-linear model at time t^n.

Formally, the present discretization could be in time only and not in space i.e. considering the differential operators semi-discretized in time only.

Data count for one 12h 4D-Var cycle
(0900-2100 UTC, 3 March 2008)

	Screened		Assimilated	
Synop:	450,000	0.3%	64,000	0.7%
Aircraft:	434,000	0.3%	215,000	2.4%
Dribu:	24,000	0.02%	7,000	0.1%
Temp:	153,000	0.1%	76,000	0.8%
Pilot:	86,000	0.1%	39,000	0.4%
AMV's:	2,535,000	1.6%	125,000	1.4%
Radiance data:	150,663,000	96.9%	8,207,000	91.0%
Scat:	835,000	0.5%	149,000	1.7%
GPS radio occult.	271,000	0.2%	137,000	1.5%
TOTAL:	155,448,000	100.0%	9,018,000	100.0%

Figure C.6: Meteorology. Observation usage. Figure extracted from Tremolet - Fisher, ECMWF, Colloquium 2009. We are still far from using all available observations.

For a sake of simplicity, in all this section, *only the initial condition* y_0 *is sough.* The cost function reads:

$$j(y_0) = \frac{1}{2}\sum_{n=0}^{N}(\mathcal{H}(y_n) - z_n)^T R^{-1}(\mathcal{H}(y_n) - z_n) + \frac{1}{2}(y_0 - y_b)^T B^{-1}(y_0 - y_b)$$

where $\mathcal{H} : \mathbb{R}^n \to \mathbb{R}^m$ is the observation operator, a-priori non-linear. R is the matrix of the covariance observation errors, B is the matrix of the covariance background errors, z_n is the observation vector at time t_n, and y_b is the background (i.e. the first guess).

Let us denote:
- $M_n = \partial_y \mathcal{M}_n(y_n)$ the linear tangent model at y_n (hence M_n^T is the adjoint operator),
- $H_n = \partial_y \mathcal{H}(y_n)$ the linear tangent observation operator at y_n.

Then the adjoint model reads:

$$(A_d)\begin{cases} \text{Given the states } \{y_k\}_k \in \mathbb{R}^n, \text{ find } \{p_k\}_k \in \mathbb{R}^n \text{ such that :} \\ p_{n-1} = M_n^T(p_n) + H_n^T R^{-1}(\mathcal{H}(y_n) - z_n) \quad n = N, ..., 1 \text{ (time steps)} \\ p_N = H_N^T R^{-1}(\mathcal{H}(y_N) - z_N) \end{cases} \tag{C.2}$$

And the gradient (with respect to the initial condition) reads:

$$\nabla j(y_0) = -p_0 \tag{C.3}$$

Let us remark that the gradient with respect to the initial condition equals the final state (at $t = 0$) of the adjoint.

C.3 The incremental VDA (4D-Var) algorithm

For very large scale problems (e.g. oceans and atmosphere in geophysics), it can be unaffordable to perform the 4D-var process for the full model resolution (i.e. the direct model including the full physics, and solved on the fine grids). Furthermore, if the assimilation is required for prediction (e.g. weather forecast), the forecast needs to be performed faster than the real time... For smaller scale problems (e.g. based on reduced models such as the 2D shallow-water equations), even if full 4d-var is possible, it remains CPU time consuming (see the forthcoming examples in river hydraulics).
An option to reduce the computational cost is to elaborate an incremental 4D-var method.

The basic idea is to combine empirically in the minimization process low-resolution and/or linearized equation terms with the full physics model.

Keeping the discrete notations above, the so-called *innovation vector* d_n is defined by :

$$\boxed{d_n = (z_n - \mathcal{H}(y_n)) \quad n \geq 0}$$

The innovation vector d_n measures at time t_n, the discrepancy between the model output and the observed quantity, in the observation space.

The basic idea of the incremental 4D-var method is to modify the 4D-var algorithm as follows:
1) the iterative control variable corrections are performed using a low-resolution model (both in physics and grids). It gives low-resolution inner-loops.
2) Once these low-resolution inner-loops have converged, the discrepancy between the original full model and the measurements (i.e. the innovative vector) is computed.
*) Repeat the process.

For a sake of simplicity, we define the operator model $\mathcal{M}$ as follows :

$$\boxed{y_n = \mathcal{M}(y_0)}$$

If we set: $y_b = y_0 + \delta y_0$, the sough quantity becomes δy_0 and we have:

$$j(y_0) = \frac{1}{2}\sum_{n=0}^{N}(\mathcal{H}(y_n) - z_n)^T R^{-1}(\mathcal{H}(y_n) - z_n) + \frac{1}{2}\delta y_0^T B^{-1}\delta y_0$$

Next, the "perturbation" δy_n corresponding to the perturbation δy_0 is defined as follows:

$$y_n + \delta y_n = \mathcal{M}(y_0 + \delta y_0)$$

A formal linearization (Taylor's expansion order 1) gives :

$$\mathcal{M}(y_0 + \delta y_0) \approx \mathcal{M}(y_0) + M(y_0) \cdot \delta y_0 \text{ hence } y_n + \delta y_n \approx y_n + M(y_0) \cdot \delta y_0$$

where M is the linear tangent model at time step t_n. Then : $\delta y_n \approx M(y_0) \cdot \delta y_0$.

Similarly:

$$\mathcal{H}(y_n + \delta y_n) \approx \mathcal{H}(y_n) + H_n \cdot \delta y_n$$

with H_n the linearized observation operator at time step t_n.

We have:

$$\boxed{j(y_0 + \delta y_0) \approx g(\delta y_0) = \frac{1}{2}\sum_{n=0}^{N}(H_n \cdot \delta y_n - d_n)^T R^{-1}(H_n \cdot \delta y_n - d_n) + \frac{1}{2}\delta y_0^T B^{-1}\delta y_0}$$

The basic idea of the incremental 4D-var method is to minimize the cost function with respect to the increment δy_0. Considering the cost function $g(\delta y_0)$, the inverse problem becomes *linear-quadratic optimal control problem* since $\delta y_n \approx M(y_0) \cdot \delta y_0$. Then the minimization can be performed using the Conjugate Gradient with preconditioning, hence very fast to compute.

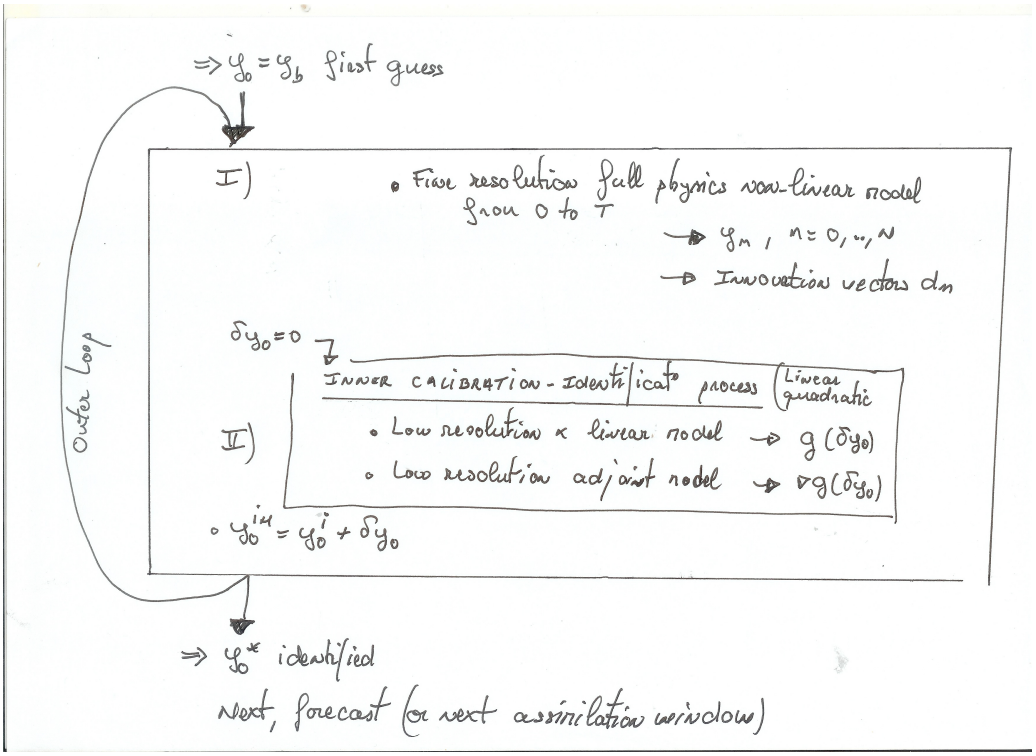


Figure C.7: The incremental VDA (4D-Var) algorithm

In summary, the 4D-var incremental algorithm reads as follows (see Figure C.7).

*) Inner loop.
The minimization process is performed with linearized operators (M and H_n), furthermore potentially with coarser grids and simplified physics.
Since the cost function $g(\delta y_0)$ is strictly convex (it is quadratic in δy_0), the extremely efficient Conjugate Gradient method with preconditioning can be used to minimize g.

*) Outer loop.
Once the low-resolution minimization process has converged (the so-called "analysis step" is done), the original "high-resolution / full" model is performed (it is the so-called "prediction stage"). The prediction stage gives new innovation vectors (the discrepancy between the model and the observations).

Let us point out that the innovation vectors d_n are the crucial input of the assimilation method, thus they are computed using the high-resolution model.

*) Update the increment and the reference state.

Thus, *the incremental 4D-var method is a mix of calibration on linearized operators, full physic predictions and discrepancy measurements based on the fine innovation vector.*

For more details, the reader should consult [10, 42], where the method is developed in a weather forecast context.

C.4 Other VDA algorithm variants

The sequential 3D-var The 3D-var algorithm is a time-independent version of the 4D-var.

Historically, it has been used for time-dependent problems because the 4D-var was too much CPU-time consuming. The 3D-var algorithm can be described as follows. During a time interval, all observations are gathered, then the discrepancy between the model and the observations is computed at a single time step (eg the final time of the interval considered). In other words, all observations are gathered (and potentially averaged) at the assimilation time. Next, the minimization process is performed.
No time integration of the direct model nor of the adjoint model have to be performed; only steady-state assimilations are performed at a given instant.
Next the unsteady model is performed until the next assimilation time.
The 3D-var drawback applied to a time-dependent model is obvious: the model outputs and the observations are not compared at the right instants...

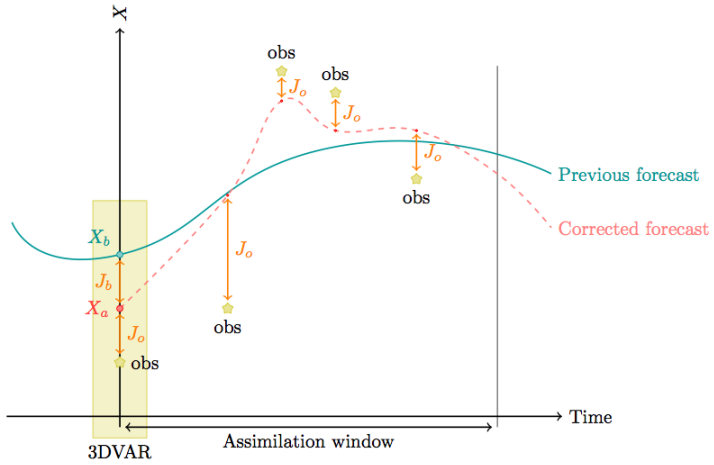
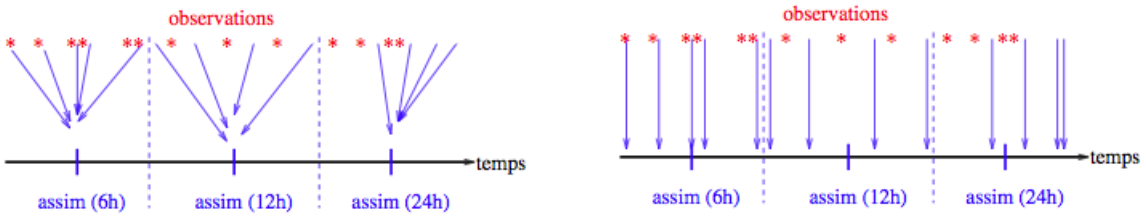


Figure C.8: Comparison of algorithms: 3D-var (sequential, time-independent) vs 4D-var (with time integration). Figure source: the ECMWF data assimilation course (www.ecmwf.int) by F. Bouttier, P. Courtier, [10].



(a) (b)

Figure C.9: Left) Algo 3D-var: During a time interval, all observations are gathered. Right) Algo 3D-FGAT: each observation is assimilated at the right instant but the linear tangent and the adjoint models are replaced by the identity operator. Image source: A. Vidard's PhD manuscript [43].

Variant: 3D-FGAT In the 3D-FGAT version (FGAT for First Guess at Appropriate Time), the linear tangent model and adjoint model are replaced by the identity operator. It is an empirical method between the 3D-var and the 4D-var. The only advantage compared to the 3D-var is the innovation vector is evaluated at the right observation times.

Other deterministic data assimilation methods

Finally, let us cite other deterministic data assimilation algorithms: the "reduced 4D-var" method and the "4D-PSAS" method are extensions of the present formalism (adjoint method, optimality system, local minimization) while the third method "Back and Forth Nudging" is not based on the present optimal control formalism.

Reduced 4D-Var consisting to elaborate order reductions (based on Empirical Orthogonal Functions EOF) may be applied to the present control problem. This may lead to a drastic reduction of the control space dimension, see e.g. [35] and references therein.

The 4D-PSAS (Physical space Statistical Analysis System), a *dual method*. It consists to set the minimization problem in the observation space instead of the state space, hence a lower-dimensional minimization problem, see [16, 4] and references therein.
In the case of a "perfect model" (no RHS error term is introduced into the direct model) then the 4D-Var and 4D-PSAS methods are equivalent. But in presence of a model error term (as a direct model source term), this term is naturally taken into account in the 4D-PSAS method. Furthermore, the dual approach performs minimization in the observation space, which is smaller than the state space.

Finally, let us cite the *Back and Forth Nudging* (BFN) method [5], which presents the advantage to circumvent the adjoint model derivation. From [5]: "The standard nudging technique consists in adding to the equations of the model a relaxation term that is supposed to force the observations to the model. The BFN algorithm consists in repeatedly performing forward and backward integrations of the model with relaxation (or nudging) terms, using opposite signs in the direct and inverse integrations, so as to make the backward evolution numerically stable".

Bibliography

[1] G. Allaire. *Numerical analysis and optimization.* Oxford university press, 2007.

[2] Mark Asch, Marc Bocquet, and Maëlle Nodet. *Data assimilation: methods, algorithms, and applications.* SIAM, 2016.

[3] Richard C Aster, Brian Borchers, and Clifford H Thurber. *Parameter estimation and inverse problems.* Elsevier, 2018.

[4] D. Auroux. Etude de differentes methodes d'assimilation de donnees pour l'environnement. *PhD thesis. Nice - Sophia-Antipolis university, France*, 2003.

[5] Didier Auroux and Jacques Blum. A nudging-based data assimilation method: the back and forth nudging (bfn) algorithm. *Nonlinear Processes in Geophysics*, 15(2):305–319, 2008.

[6] Combettes P.-L. Bauschke H. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces.* Springer, 2011.

[7] A. F. Bennett. *Inverse modeling of the ocean and atmosphere.* Cambridge University Press, 2005.

[8] J. Blum, F.-X. Le Dimet, and Navon I.M. Data assimilation for geophysical fluids. In Temam and Tribbia, editors, *Handbook of Numerical Analysis*, volume 14. North-Holland, 2009.

[9] J.F. Bonnans, J.C. Gilbert, C. Lemaréchal, and C.A. Sagastizábal. *Numerical optimization: theoretical and practical aspects.* Springer, 2006.

[10] L. Bouttier and P. Courtier. *Data assimilation concepts and methods.* ECMWF Training course. www.ecmwf.int., 1999.

[11] H. Brézis. *Functional Analysis, Sobolev Spaces and Partial Differential Equations.* Springer, 2010.

[12] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *PNAS*, 113(15), 04 2016.

[13] Emmanuel Candès and Justin Romberg. Sparsity and incoherence in compressive sampling. *Inverse problems*, 23(3), 04 2007.

[14] Alberto Carrassi, Marc Bocquet, Laurent Bertino, and Geir Evensen. Data assimilation in the geosciences: An overview of methods, issues, and perspectives. *Wiley Interdisciplinary Reviews: Climate Change*, 9(5):e535, 2018.

[15] P. Courtier and O. Talagrand. Variational assimilation of meteorological observations with the direct adjoint shallow water equations. *Tellus*, 42(A):531–549, 1990.

[16] Philippe Courtier. Dual formulation of four-dimensional variational assimilation. *Quarterly Journal of the Royal Meteorological Society*, 123(544):2449–2461, 1997.

[17] DassFlow. "data assimilation for free surface flows". Open source computational software: http://www.math.univ-toulouse.fr/DassFlow.

[18] Lawrence C. Evans. *An Introduction to Mathematical Optimal Control Theory*. University of California, Berkeley, 2014.

[19] Geir Evensen. *Data assimilation: the ensemble Kalman filter*. Springer Science & Business Media, 2009.

[20] A. Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Number 19 in Frontiers in Appl. Math. SIAM, Philadelphia, PA, 2000.

[21] Matthew CG Hall and Dan G Cacuci. Physical interpretation of the adjoint functions for sensitivity analysis of atmospheric models. *Journal of the atmospheric sciences*, 40(10):2537–2546, 1983.

[22] L. Hascoet. *Automatic Differentiation by Program Transformation*. INRIA Sophia-Antipolis, TROPICS team. Available on-line., 2007.

[23] L. Hascoet, V. Pascual, et al. *Tapenade software*. INRIA, project Tropics, www-sop.inria.fr/tropics, 2012.

[24] M. Honnorat. Assimilation de données lagrangiennes pour la simulation numérique en hydraulique fluviale. *PhD thesis. Grenoble IT, France*, 2007.

[25] Barbara Kaltenbacher, Andreas Neubauer, and Otmar Scherzer. *Iterative regularization methods for nonlinear ill-posed problems*, volume 6. Walter de Gruyter, 2008.

[26] Andreas Kirsch. *An introduction to the mathematical theory of inverse problems*, volume 120. Springer Science & Business Media, 2011.

[27] F.X. Le Dimet and O. Talagrand. Variational algorithms for analysis assimilation of meteorological observations: theoretical aspects. *Tellus A*, 38:97–110, 1986.

[28] J.L. Lions. *Contrôle optimal de systèmes gouvernés par des équations aux dérivées partielles*. Dunod, 1968.

[29] J.L. Lions. *Optimal control of systems governed by partial differential equations*. Springer-Verlag, 1971.

[30] GI Marchuk and VP Shutyaev. Iteration methods for solving a data assimilation problem. *Russian Journal of Numerical Analysis and Mathematical Modelling*, 9(3):265–280, 1994.

[31] GI Marchuk and VB Zalesny. A numerical technique for geophysical data assimilation problems using pontryagin?s principle and splitting-up method. *Russian Journal of Numerical Analysis and Mathematical Modelling*, 8(4):311–326, 1993.

[32] J. Monnier. Modèles numériques directs et inverses d'écoulements de fluides. *Habilitation à Diriger des Recherches. Institut National Polytechnique de Grenoble (Grenoble IT)*, 2007.

[33] J. Monnier, K. Larnier, P. Brisset, and F. et al. Couderc. Dassflow (data assimilation for free surface flows): numerical analysis report. Open source computational software: http://www.math.univ-toulouse.fr/DassFlow.

[34] U. Naumann. *The Art of Differentiating Computer Programs: An Introduction to Algorithmic Differentiation*. SIAM, 2012.

[35] Céline Robert, Eric Blayo, and Jacques Verron. Comparison of reduced-order, sequential and variational data assimilation methods in the tropical pacific ocean. *Ocean Dynamics*, 56(5-6):624–633, 2006.

[36] Samuel H. Rudy, Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Data-driven discovery of partial differential equations. *Science Advances*, 3(4), 04 2017.

[37] Yoshikazu Sasaki. An objective analysis based on the variational method. *J. Meteor. Soc. Japan*, 36(3):77–88, 1958.

[38] Yoshikazu Sasaki. Some basic formalisms in numerical variational analysis. *Monthly Weather Review*, 98(12):875–883, 1970.

[39] O. Thual. *Hydrodynamique de l'environnement*. Editions de l'Ecole Polytechnique, 2010.

[40] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1), 1996.

[41] E. Trélat. *Optimal control: theory and applications*. Vuibert, Paris, 2008.

[42] Y. Trémolet. Incremental 4d-var convergence study. *Tellus A*, 59(5):706–718, 2008.

[43] A. Vidard. Vers une prise en compte de l'erreur modèle en assimilation de données 4d-variationnelle. *PhD thesis. Grenoble university J. Fourier, France*, 2001.

[44] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 67(2), 2005.