Lecture 9: Large-scale optimization for function estimation

Y. Favennec[1] **and P. Le Masson** [2]
[1] LTN – UMR CNRS 6607 – Polytetch' Nantes – 44306 Nantes – France
E-mail: Yann.Favennec@univ-nantes.fr
[2] LIMATB – Université de Bretagne Sud – 56325 Lorient – France

Abstract. This lecture presents some commonly-used numerical algorithms devoted to optimization, that is maximizing or, more often minimizing a given function of several variables. The goal is function estimation. At first, some general mathematical tools are presented. Some gradient-free optimization algorithms are presented and then some gradient-type methods are pointed out with pros and cons for each method. The function to be minimized gradient is presented accordingly to three distinct methods: finite difference, forward differentiation and the use of the additional adjoint problem. The last part presents some practical studies where some tricks are given along with some numerical results.

Keywords. Optimization, Convexity, Zero-order method, Deterministic method, Stochastic method, Gradient-type method, Conjugate gradient, BFGS, Gauss–Newton, Levenberg–Marquardt, Gradients, Direct differentiation, Adjoint-state

---

Contents

1 Introduction

This lecture is devoted to the use of optimization methods for the solution of non-linear large-scale inverse problems in the field of heat transfer.
The lecture first presents some basic examples of IHCP (Inverse Heat Conduction Problems) and points out the distinction between estimation of parameters and functions. Typically, one presents the difference between estimating λ as a parameter, $\lambda(\boldsymbol{x})$ as a function of the space $\boldsymbol{x}$ ($\boldsymbol{x} = (x_1, x_2)^t$ for instance), and $\lambda(T)$ as a function of the state T.
The lecture then presents the most usual optimization tools for the solution of different kinds of inverse problems. It first gives notions on the functional to be minimized, and convexity. It gives

definitions of constraints (equality and inequality) added to the functional to be minimized, the added constraints being related to either the state or the parameter/functional.
Then, before going into details on the iterative optimization algorithms, the most usual stopping criteria are presented.
Zero-order, first-order and quasi-second order optimization methods are briefly presented with pros and cons for each of them.
Concerning zero order methods, both deterministic and stochatic methods are very briefly presented with some specific examples (Simplex, PSO, and GA). This part is to be related to the Metti tutorial devoted to"zero-order optimization".
Within the frame of first-order methods, one presents the steepest-descent method with and without line-search, then the conjugate gradient method for quadratic and arbitrary functions.
Some quasi-Newton algorithms are then presented: the BFGS, the Gauss–Newton and the Levenberg–Marquardt methods.
A comparison is given in terms of gradient needed for all previously presented method along with convergence rate if possible.
The next part presents the computation of the functional gradient: through the finite difference method, through the direct differentiation of the PDEs (partial differential equations), and through the use of the adjoint-state problem. Several ways to access the adjoint-state problems are given. A comparison of gradient computation is given on examples to emphasize the differences.
Note that this lecture has been prepared with some well-known books such that [1, 2, 3, 4, 5]. These books being considered as "standard" famous books, some parts of this lecture are taken from these references. Moreover, this lecture is actually an improvement of previous Metti lectures devoted to optimization [6, 7].

2 Estimation in heat transfer – Optimization

2.1 Parameter and function estimation

The modelling of a physical system leans on several requirements. Added to the physical modelling equations that include some physical parameters (e.g. conductivity coefficients), the initial state and the sources are also to be known if the physical problem is to be solved. If all this data is known, then the "forward" problem can be solved: this is the so-called "classical" physical modelling.

Now if some of the previously expressed quantities are missing, the physical problem cannot be solved any longer, but some inversion procedure may evaluate the missing quantity so that the model output fits with some real ones (i.e. obtained through experiments).

There was these last years a debate within the heat transfer community about the difference and the meaning of, on one hand, "parameter identification" and, on the other hand, "function estimation". According to the lecture's authors, both are almost the same, at least for the solution procedure, even though, initially, some differences may be expressed.

Let us work on the example of a conductivity estimation to back up our methodology.

- If a material conductivity λ (scalar or tensor) is to be identified, then a parameter identification is to be dealt with. Moreover in this case, the number of unknown "parameters" is very low, so that specific algorithms will be used (e.g. gradient free optimizers or matrix-

based optimizers such as the Gauss–Newton method with the computation of the gradient through direct differentiation).

- If the physical parameter now depends continuously on the state, (e.g. thermo-dependant conductivity λ), then one sould identify the function $\lambda = \lambda(T)$. Creating a projection basis of the form $\lambda = \sum \lambda_i \xi^i(T)$ where the λ_i are to be evaluated, then a parameter estimation is eventually also dealt with.

- If the physical property depends on $\boldsymbol{x}$, the function $\lambda(\boldsymbol{x})$ is to be parameterized through $\lambda = \sum \lambda_i \xi^i(\boldsymbol{x})$ and then, as above, one searches the λ_i. At this stage, one finally deals with parameter estimation. The basis $\xi^i(\boldsymbol{x})$ may be a finite element basis for instance if this one is used for searching the state solution, or any other basis such as polynomial, spline functions, etc.

Actually, one usually speaks of function estimation when the quantity to be evaluated depends on the state, the location $\boldsymbol{x}$, or the time t. Nevertheless, in final, the function to be estimated is usually discretized, that is parameterized, so that we turn out to be back to parameter estimation.

A commonly accepted difference between what is called parameter and function estimation is that in the first case there is usually few parameters to be estimated (say less that 100 – though this is not the case in model reduction for instance while it is for sure parameter estimation, and in the second case there is usually a large number of parameters to be estimated. Anyway, for the second case, one has to keep in mind that usually the parameterization must not be "as finer as possible" for regularity considerations.

Even though function estimation can sometimes be treated in the same way as parameter estimation, a great difference between both comes from regularization because one can profit that $\lambda(\boldsymbol{x})$ is indeed a function of $\boldsymbol{x}$ for instance so that one can build specific penalization functions of $\boldsymbol{x}$ before being discretized, or even specific inner products. Such approaches cannot be built on parameters by themselves. Several specific regularizations used for large-scale functions are given in Section 8.

2.2 The function to be minimized

In inversion process, one usually minimizes some errors between some experimental data (say u_d noted y in some other Metti Lectures) and related model data (say u noted y_{mo} in some other Metti Lectures). The cost function (also called somewhere discrepancy function or objective function) is often expressed as a norm of the difference between u and u_d. The most often, one uses the $L_2(\cdot)$ norm if some "quasi-"continuous u and especially u_d are available (i.e. $\|u-u_d\|^2_{L_2(\mathscr{S})} = \int_{\mathscr{S}} (u-u_d)^2 \,\mathrm{d}\boldsymbol{x}$ but, when data u_d is given only on specific locations (in space and/or time) , then the squared euclidean norm is to be used: $\|u-u_d\|_2^2 := \sum_i (u(\boldsymbol{x}_i) - u_d(\boldsymbol{x}_i))^2 = \int_{\mathscr{S}} \delta_i^j (u-u_d)^2 \,\mathrm{d}\boldsymbol{x}$ where $\delta_i^j = \delta(\boldsymbol{x}^i - \boldsymbol{x}^j)$. Often, some function of the state and of the measure are used, for instance state derivation, integration, weighted summation, etc. Moreover, some selection process is, most of the time considered. So, in order to write down a general form for the cost function to be minimized, we use :

$$\mathscr{J}(u) = \|u-u_d\|^2_{\mathscr{X}} \tag{1}$$

without specifying any choice for the norm on $\mathscr{X}$ at this early stage. The norm is squared so that the function $\mathscr{J}$ does not *a priori* present any discontinuity. Though the cost function is explicitly given in terms of the state u, the cost function is actually to be minimized with respect to what it is searched, i.e. the parameters ψ. Hence we write the equality (by definition):

$$j(\psi) := \mathscr{J}(u,\psi) \tag{2}$$

where the function j is often called the reduced cost function, as opposed to $\mathscr{J}$ which is the calculated cost function. One actually computes the cost function in terms of the state (by (1) for instance), but the cost function is to be minimized with respect to another quantity, say ψ.

2.3 Elements of minimization

The function denoted j in (2) is defined on $\mathscr{K}$ with values in $\mathbb{R}$. $\mathscr{K}$ is a set of admissible elements of the problem. In some cases, $\mathscr{K}$ defines some constraints on the parameters or functions. The minimization problem is written as:

$$\inf_{\phi\in\mathscr{K}\subset\mathscr{V}} j(\phi).$$

According to [1], if the notation "inf" is used for a minimization problem, it means that one does not know, a priori, is the minimum is obtained, i.e. if there exists $\phi\in\mathscr{K}$ such that

$$j\left(\phi\right) = \inf_{\psi\in\mathscr{K}\subset\mathscr{V}} j(\psi).$$

For indicating that the minimum is obtained, one should prefer the notations

$$\phi = \arg\min_{\psi\in\mathscr{K}\subset\mathscr{V}} j(\psi) \text{ and } j(\phi) = \min_{\psi\in.\mathscr{K}\subset\mathscr{V}} j(\psi)$$

Let us now recall basic definitions needed for mathematical optimization [1]:

Definition 1. *ψ is a local minimum of j on $\mathscr{K}$ if and only if*

$$\psi\in\mathscr{K} \text{ and } \exists\delta>0,\ \forall\phi\in\mathscr{K}, \|\phi-\psi\|<\delta \ \rightarrow\ j(\phi)\geq j(\psi).$$

Definition 2. *ψ is a global minimum of j on $\mathscr{K}$ if and only if*

$$\psi\in\mathscr{K} \text{ and } j(\phi)\geq j(\psi)\ \forall\phi\in\mathscr{K}.$$

Definition 3. *A minimizing series of j in $\mathscr{K}$ is a series $\left(\psi^n\right)_{n\in\mathbb{N}}$ such that*

$$\psi^n\in\mathscr{K}\ \forall n \text{ and } \lim_{n\rightarrow+\infty} j\left(\psi^n\right) = \min_{\phi\in.\mathscr{K}} j(\phi).$$

Definition 4. *a set $\mathscr{K}\in\mathscr{V}$ is convex if, for all $\psi,\phi\in\mathscr{K}$ and $\forall\theta\in[0,1]$, the element $(\theta\psi+(1-\theta)\phi)$ is in $\mathscr{K}$ (see figure 1).*

Definition 5. *A function j is said to be convex when defined on a non-null convex set $\mathscr{K}\in\mathscr{V}$ with values in $\mathbb{R}$ if and only if*

$$j\left(\theta\psi+(1-\theta)\,\phi\right)\leq\theta\, j\left(\psi\right)+(1-\theta)\, j\left(\phi\right)\ \forall\psi,\phi\in\mathscr{K},\ \forall\theta\in[0,1].$$

Moreover, j is said to be strictly convex if the inequality is strict when $\psi\neq\phi$ and $\theta\in\,]0,1[$ (see figure 2).

Ending, if j if a convex function on $\mathscr{K}$, the local minimum of j on $\mathscr{K}$ is the global minimum on $\mathscr{K}$.

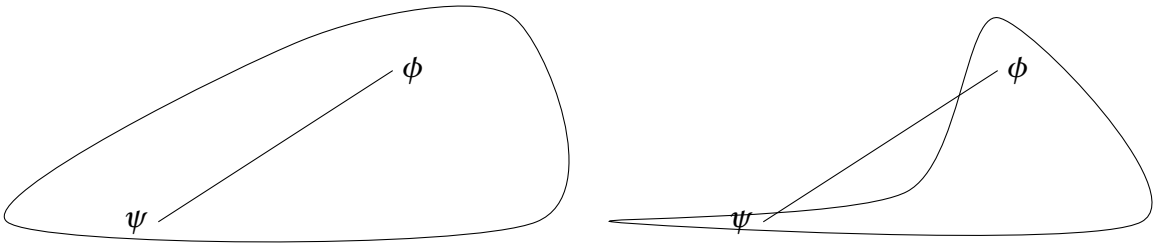


Figure 1: Convex and non-convex domaine $\mathcal{K}$

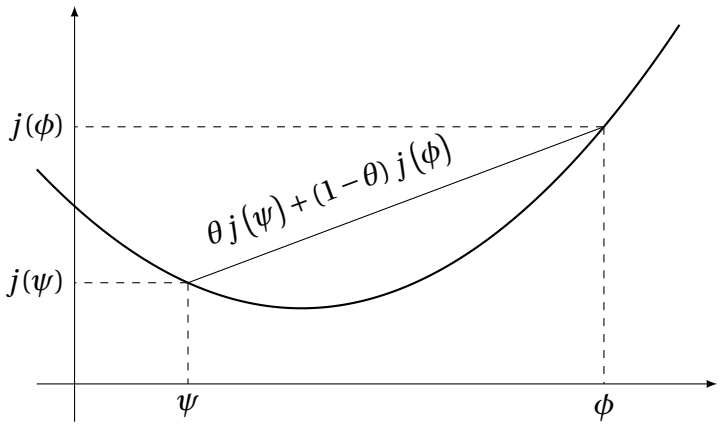


Figure 2: Convex function $j(\cdot)$

2.4 Optimality conditions

For convex functions, there is no difference between local minima and global minimum. In the following we are more interested in minimizing a function without specifying whether the minimum is local or global. It will be seen in next sections that some gradient-free optimization algorithms may find the global minimum even if the cost function contains local minima.

Let us derive here the minimization necessary and sufficient conditions. These conditions use the first-order derivatives (order-1 condition), and second-order derivatives (order-2 condition) on the cost function j. Using gradient-type algorithms, the first–order condition is to be reached, while the second-order condition leads to fix the convexity hypothesis, and then make a distinction bewteen minima, maxima and optima.

Let us assume that $j(\psi)$ is continuous and has continuous partial first derivatives $(\partial j/\partial \psi_i)(\psi)$ and second derivatives $(\partial^2 j/\partial \psi_i \partial \psi_j)(\psi)$. Then a *necessary condition* for $\bar{\psi}$ to be a minimum of j (at least locally) is that:

i) $\bar{\psi}$ is a stationary point, i.e. $\nabla j(\bar{\psi}) = 0$,

ii) the Hessian $\nabla^2 j(\bar{\psi}) = \left(\partial^2 j/\partial \psi_i \partial \psi_j\right)(\bar{\psi})$ is a positive semi-definite matrix, i.e. $\forall y \in \mathbb{R}^n$, $\left(\nabla^2 j(\bar{\psi})\, y, y\right) \geq 0$ where $(.,.)$ is a scalar product in $\mathbb{R}^n$ (we have $\dim(\psi) = n$).

A point $\bar{\psi}$ which satisfies condition i) is called a *stationary point*. It is important to point out that stationarity is not a sufficient condition for local optimality. For instance the point of

inflexion for cubic functions would satisfy condition i) while there is no optimum. Hence the Hessian is not positive definite but merely positive semi-definite.

The *sufficient condition* for $\bar{\psi}$ to be a minimum of j (at least locally) is that

i) $\bar{\psi}$ is a stationary point, i.e. $\nabla j(\bar{\psi}) = 0$,

ii) the Hessian $\nabla^2 j(\bar{\psi}) = \left(\partial^2 j / \partial\psi_i \partial\psi_j\right)(\bar{\psi})$ is a positive definite matrix, i.e. $\forall y \in \mathbb{R}^n,\ y \neq 0$, $\left(\nabla^2 j(\bar{u})\, y, y\right) > 0$.

We remark that condition ii) amounts to assuming that j is strictly convex in the neighbourhood of $\bar{\psi}$.

2.5 Stopping criteria

Since the convergence of the iterative algorithms is, in general, not finite, a stopping criterion must be applied. Here below are given some commonly used criteria. We denote ψ^p the vector parameter ψ at the optimization iteration p.

$$\left\|\nabla j(\psi^p)\right\| \leq \varepsilon; \tag{3}$$

$$\left|j(\psi^p) - j(\psi^{p-1})\right| \leq \varepsilon; \tag{4}$$

$$\left\|\psi^p - \psi^{p-1}\right\| \leq \varepsilon; \tag{5}$$

$$j(\psi^p) \leq \boldsymbol{v} \tag{6}$$

For each of the above criteria, it may be judicious to demand that the test be satisfied over several successive iterations. The three first above-presented criteria are convergence criteria applied on the cost function gradient, on the cost function evolution, or on the parameter evolutions. These criteria are very commonly-used when dealing with optimization and optimal control problems.

The last criterion is, in one sense, more specific to inverse problems: when the cost function reaches a critical value that depends on the variance of measurement errors, then the optimization algorithm should stop [8]. It can be shown that the consequence of lowering the cost function below $\boldsymbol{v}$ affects the result in dramatically highliting its inherent noise. This criterion is the "maximum discrepancy principle".

Often, the maximum discrepancy principle (6) is used together with the other criteria and also with a maximum number of iterations.

2.6 Classification of optimization methods

The solution of the optimization problem may be performed in numbers of ways. Among numerous methods found in the litterature, the classification of methods given below (see Figure 3) is based on our experience. First, one can distinguish gradient-free methods from methods relying on gradients. Among gradient-free methods, there are those deterministic and those stochastic (the latter introducing random in the search of optimum). Among gradient-based methods, one can distinguish between first and second-order methods and those in between.

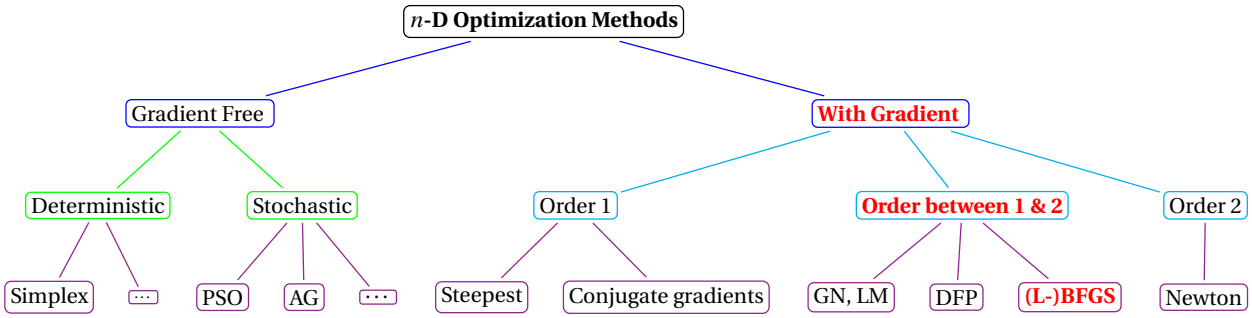


Figure 3: Classification of optimization methods

3 Zero-order n–dimensional optimization algorithms

Zero-order methods, also called "derivative-free optimization" (DFO) or "gradient-free methods" are based on a global vision of the cost function value j on the search space. The main interest of using such methods is when the cost function gradient is not available, or when the cost gradient is not easy to compute, or when the cost function presents local minima. There is an increasing number of computation tools to solve optimization problems with no gradient [9]. In the sequel, we restrict ourself in very briefly presenting, among the enormous number of existing methods, one deterministic algoritm which is the so-called simplex method, and one probabilistic method which is the particle swarm optimization method.

3.1 Simplex

We present here the Nelder-Mead simplex method (1965). This method is popular and simple to code. Moreover, there exists a large number of freeware that can be used to minimize a function using such algorithm. Let a simplex $\mathscr{S}_0$ be a set of $n+1$ "points" linearly independant ($n = \dim \psi$) with $\mathscr{S}_0 = \{\psi^I, I = 1,\dots,n+1\}$. One iteration of the simplex optimization algorithm consists in generating a new simplex closer to the minimum eliminating the point with the higher cost function value. The basic operations of $n = 2$ are given in Figure 4). Let $\bar{\psi}$ the isobarycenter of $\{\psi^I, I = 1,\dots,n,\}$ (without ψ^h), let the ordering so that

$$j(\psi^1) \leq j(\psi^2) \leq \ldots \leq j(\psi^{n+1})$$

and let $\psi^\ell = \arg_{I=1,\dots,n} \min j(\psi^I)$ and $\psi^h = \arg_{I=1,\dots,n} \max j(\psi^I)$. At each iteration, the simplex improvement is performed in three steps:

1. [Reflexion] One builds ψ^R symetry of ψ^h with respect to the segment $[\bar{\psi}, \psi^\ell]$. According to the value of the cost $j(u^R)$ with respect to $j(\psi^\ell)$, the parametric space is then extended (step 2) or contracted (step 3);

2. [Extension] if $j(\psi^R) < j(\psi^\ell)$, one searches a new point in the same direction. The point ψ^E is such that $\psi^E = \gamma\psi^R + (1-\gamma)\bar{\psi}$ with $\gamma > 1$. If $j(\psi^E) < j(\psi^R)$, ψ^h is replaced by ψ^R, otherwise ψ^h is replaced by ψ^E;

3. [Contraction] If $j(\psi^R) > j(\psi^\ell)$, the point ψ^C such that $\psi^C = \gamma\psi^h + (1-\gamma)\bar{\psi}$, $\gamma \in]0,1[$ is created. If $j(\psi^C) < j(\psi^R)$, ψ^h is replaced by ψ^C otherwise the simplex is contracted (inside contraction) in all directions replacing $\forall I \neq L\ \psi^I$ by $(\psi^I + \psi^\ell)/2$.

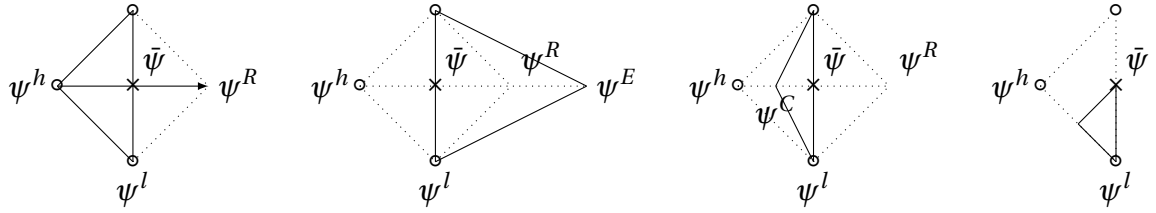


Figure 4: Basic operations on a simplex for $n = 2$. From left to right: reflection, expansion, contraction and inside contraction.

3.2 PSO

The particle swarm optimization is a stochastic algorithm described by Kennedy and Eberhart in 1995. One considers an initial set of individuals (particles) located randomly. Each particle moves within the space $\mathscr{K}$ interacting with other particles on their best locations. From this information, the particle shall change its position ψ^i and its velocity $\delta\psi^i$. The general formulation for this behavior is given by:

$$\begin{aligned} \delta\psi^i &= \chi\delta\psi^i + \lambda_1 \mathrm{rand}_1\left(\phi^g - \psi^i\right) + \lambda_2 \mathrm{rand}_2\left(\phi^i - \psi^i\right) \\ \psi^i &= \psi^i + \delta\psi^i \end{aligned} \tag{7}$$

where ψ^i is the position of the particle i, $\delta\psi^i$ is its velocity, ϕ^g is the best position obtained in its neighborhood, and ϕ^i is its best position (see Figure 5). χ, λ_1 and λ_2 are some coefficients weighting the three directions of the particule [9]:

- how much the particle trusts itself now,
- how much it trusts its experience,
- how much it trusts its neighbours.

Next, rand_1 and rand_2 are random variables following a uniform distribution in $[0,1]$. There are several configuration parameters for the method, see [10]:

- swarm size, usually between 20 and 30;
- initialization of both the position of the particles and their velocity $\sim \mathscr{U}[0,1]$,
- neighborhood topology such that a particule communicates with only some other particles,
- inertial factor χ which defines the exploration capacity of the particules,
- confidence coefficients λ_1 and λ_2 which are constriction coefficients,
- stopping criterion which is usually the maximum of iterations, or the critical value of the cost function $j(\psi)$.

Usually, a circular neighborhood topology is used, along with $\chi = 0.72$ and $\lambda_1 = \lambda_2 = 1.46$. A large number of free software are available, see for instance [11].

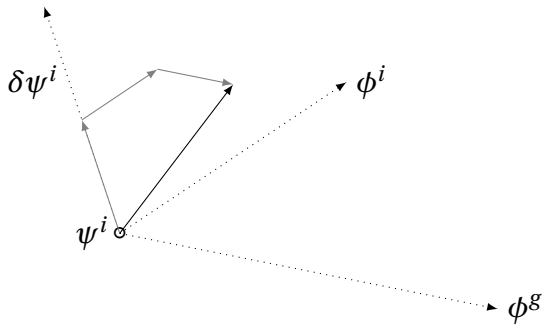


Figure 5: PSO algorithm: a particle displacement.

4 One-dimensional unconstrained opimization – Line search algorithm

In order to find the optimum of a function j of n variables, we shall describe in section 5 a number of iterative methods which require, at each step, the solution of an optimization problem in one single variable, of the type:

$$\text{Find } \bar{\alpha} = \arg\min_{\alpha>0} g(\alpha) = j\left(\psi^p + \alpha d^p\right), \tag{8}$$

where $\psi^p = \left(\psi_1^p \dots \psi_n^p\right)^t$ is the obtained point at iteration p and where $d^p = \left(d_1^p \dots d_n^p\right)^t$ is the direction of descent (see section 5). As a matter of fact we have the problem of finding the optimum of the function j, starting from the guess ψ^0 in the direction of descent d^0. Since this problem must be solved a great number of times, it is important to design efficient algorithms that deal with it. In any case, one has to keep in mind that the main objective is not to solve (8) but to find the minimum of $j(\psi)$. Thus one has to design efficient tools for the one-dimensional algorithm that finds the minimum of $g(\alpha)$ or approach it, in a not so expensive way. Note that we always assume that $g'(0) = \left(\nabla j(\psi^p), d^p\right) < 0$ meaning that d^p is indeed a descent direction.

4.1 The dichotomy method

This method halves, at each step, the length of the interval which contains the minimum, by computing the function g in two new points. By carrying out n computations of the function g, the length of the initial interval $[a^0, b^0]$ is reduced in a proportion of $2^{(n-3)/2}$. The general procedure is the following. Starting from the interval $[a^0, b^0]$ and taking the midpoint $c^0 = \left(a^0 + b^0\right)/2$ and the two points $d^0 = \left(a^0 + c^0\right)/2$ and $e^0 = \left(c^0 + b^0\right)/2$ one obtains five equidistant points of length $\delta^0 = (b^0 - a^0)/4$. Computing the cost function values at these points, two of the four subintervals may be eliminated while two adjacent subintervals remain. The same procedure is repeated within the selected interval $[a^1, b^1]$ and so on. Since the step length is divided by 2 at each iteration, the dichotomy method converges linearly to the minimum [2].

4.2 The Newton–Raphson method

Let us assume that the function $g(\alpha)$ is twice continuously differentiable. The search for a minimum of $g(\alpha)$ is carried out by looking for a stationary point, i.e. $\bar{\alpha}$ satisfying the possibly non-

linear relationship $g'(\bar{\alpha}) = 0$. If α^q is the point obtained at stage q, then the function $g'(\alpha)$ is approximated by its tangent, and the next point α^{q+1} is chosen to be at the intersection of this tangent with the zero-ordinate axis. The relationship to pass from one step to the next comes from $g'\left(\alpha^{q+1}\right) = g'\left(\alpha^q\right) + g''\left(\alpha^q\right) \times \left(\alpha^{q+1} - \alpha^q\right) = 0$ which gives:

$$\alpha^{q+1} = \alpha^q - \frac{g'\left(\alpha^q\right)}{g''\left(\alpha^q\right)}. \tag{9}$$

It is of interest that this method has the property of finite convergence when applied to quadratic functions. This is an interesting feature because any function which is sufficiently regular (at least twice continuously differentiable) behaves as a quadratic function near the optimum [2]. On the other hand, the main drawback of this method is that it requires the computation of the first and of the second derivative of g at each stage. That is the reason why the secant method (next section) is also widely used, especially when there is no way for computing the second order derivative, or when the exact second derivative is complicated to compute or too time consuming.

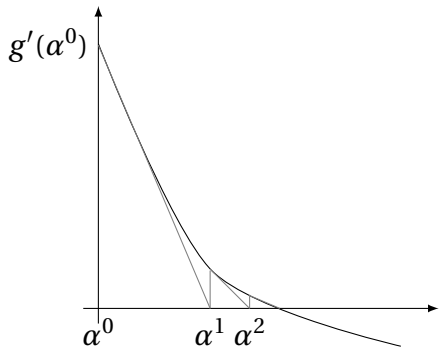


Figure 6: The Newton–Raphson line search method.

4.3 The secant method

The second-order derivative $g''(\alpha)$ is approximated by finite differences so that the Newton–Raphson's equation initially given by (9) becomes (10):

$$\alpha^{q+1} \approxeq \alpha^q - g'(\alpha^q)\frac{\alpha^q - \alpha^{q-1}}{g'\left(\alpha^q\right) - g'\left(\alpha^{q-1}\right)}. \tag{10}$$

This method is the so-called *secant method*. Applied to the search of $g'(\alpha) = 0$ this method consists in searching the intersection between the zero-ordinate axis and the straight line passing by the points $\left[\alpha^{q-1}, g'(\alpha^{q-1})\right]$ and $\left[\alpha^q, g'(\alpha^q)\right]$.

4.4 The quadratic interpolation

By comparison of those of sections 4.2 and 4.3, this method has the advantage of not requiring the computation of first or second order derivatives of the function . Let three points $\alpha_1 \leq \alpha_2 \leq \alpha_3$ such that $g(\alpha_1) \geq g(\alpha_2)$ and $g(\alpha_3) \geq g(\alpha_2)$ and let us approximate the function g on the

related interval by a quadratic function $\tilde{g}$ with the same values as g has at the points α_1, α_2 and α_3. The minimum of $\tilde{g}$ is obtained at the new point α_4 satisfying:

$$\alpha_4 = \frac{1}{2}\,\frac{r_{23}g(\alpha_1) + r_{31}g(\alpha_2) + r_{12}g(\alpha_3)}{s_{23}g(\alpha_1) + s_{31}g(\alpha_2) + s_{12}g(\alpha_3)}, \tag{11}$$

where $r_{ij} = \alpha_i^2 - \alpha_j^2$ and $s_{ij} = \alpha_i - \alpha_j$. This procedure may be repeated again with the three new selected points. Under some regularity hypothesis, this method convergence rate is superlinear [2].

Another approach consists in differentiating the cost function towards the direction of descent with Taylor development, and neglecting second order derivatives:

$$g'(\alpha) = \frac{d}{d\alpha}\left\| u\left(\psi^k + \alpha d^k\right) - u_d\right\|^2_{\mathscr{X}} = \frac{d}{d\alpha}\left\| u\left(\psi^k\right) + \alpha u'\left(\psi^k; d^k\right) - u_d\right\|^2_{\mathscr{X}} = 0 \tag{12}$$

with $u'\left(\psi, d^k\right) = u'$ the derivative of u at the point ψ^k and towards d^k. This equation gives straitforwardly:

$$\left(u', u - u_d\right)_{\mathscr{X}} + \alpha\left(u', u'\right)_{\mathscr{X}} = 0 \tag{13}$$

$$\alpha = \frac{\left(u', u - u_d\right)_{\mathscr{X}}}{(u', u')_{\mathscr{X}}} \tag{14}$$

This latter method which is widely used in the heat transfer community can give easily an accurate step size α when the cost function j is close to be quadratic, i.e. when the state u varies almost linearly with ψ.

4.5 Other methods – Inexact line-search

A great number of other one-dimensional optimization methods may be found in the literature. These methods may be more or less complicated and some of them may be much more optimal than the above-presented methods. In practice the Fibonacci method,the golden section search method and the cubic interpolation method are also very widely used in practice (the reader may refer to [5, 3, 2] for more details). All these methods can be quite CPU-time consuming, and in fact, the convergence of some of the methods presented afterwards in Section 5 (typically the BFGS method) can be reached without getting a point very close to satisfying $g(\alpha) = 0$. Well-accepted conditions used to build inexact line-search algorithms are based on the two rules:

a) α must not be too large in order, for instance, to avoid oscillations,

b) α must not be chosen too small in order to prevent from premature convergence.

Among the large number of inexact line-search algorithms, one is based on the Goldstein rules (see Figure 7) which first ensures condition a) by satisfying (15) choosing $m_1 \in [0, 1]$, and second ensures condition b) satisfying (16) choosing $m_2 \in [m_1, 1]$.

$$g(\alpha) \leq g(0) + m_1\alpha g'(0) \tag{15}$$

$$g(\alpha) \geq g(0) + m_2\alpha g'(0) \tag{16}$$

Other rules can be stated in similar ways. For instance, the Armijo's method is a variant of the Golstein method. Related algorithms are very simple and can be found in any book on optimization.

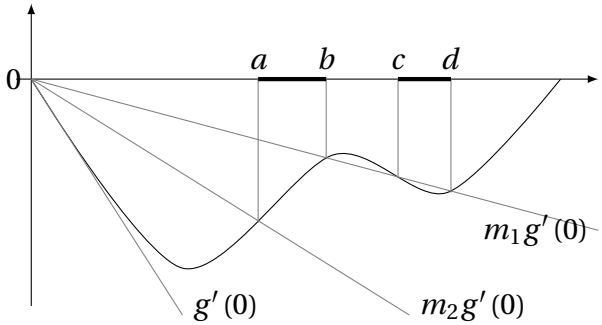


Figure 7: Set of points satisfying the Goldstein's rules: $[a,b] \cup [c,d]$.

Algorithm 1: Typical method based on the Goldstein rules

input : $\alpha_{\min} = 0$, $\alpha_{\max} = \infty$, $\psi = \psi^k$, ∇j, $d = d^k$
output: $\bar{\alpha}$

1 Give some initial value to α;
Compute $g'(0) = \left(\nabla j, d\right)$;
2 Compute $g(\alpha) = j(\psi + \alpha d)$;
if $g(\alpha) \leq g(0) + m_1 \alpha g'(0)$ **then**
| go to 3)
else
| set $\alpha_{\max} = \alpha$ and go to 5
end
3 Compare $g(\alpha)$ and $g(0) + m_2 \alpha g'(0)$;
if $g(\alpha) \geq g(0) + m_2 \alpha g'(0)$ **then**
| END
else
| go to 4
end
4 Set $\alpha_{\min} = \alpha$;
5 Look for new value in $]\alpha_{\min}, \alpha_{\max}[$ and return to 2

5 Gradient-type n-dimensional optimization algorithms

Since in all cases, the stationarity of j is a necessary optimality condition, almost all unconstrained optimization methods consist in searching the stationary point $\bar{\psi}$ where $\nabla j(\bar{\psi}) = 0$. The usual methods are iterative and proceed this way: one generates a sequence of points ψ^0, ψ^1,... ψ^p which converges to a local optimum of j. At each stage p, ψ^{p+1} is defined by $\psi^{p+1} = \psi^p + \alpha^p d^p$ where d^p is a displacement direction which may be either the opposite of the gradient of j at ψ^p (i.e. $d^p = -\nabla j(\psi^p)$), or computed from the gradient or chosen in another way provided that it is a descent direction, i.e. satisfying $\left(\nabla j(\psi^p), d^p\right) < 0$.

5.1 1st order gradient methods

5.1.1 The gradient with predefined steps method (1st order method)

At each iteration step p, the gradient $\nabla j(\psi^p)$ gives the direction of the largest increase of j. The procedure consists thus in computing the gradient, and in finding the new point according to the predefined strictly positive step size α^p as:

$$\psi^{p+1} = \psi^p - \alpha^p \frac{\nabla j(\psi^p)}{\left\| \nabla j(\psi^p) \right\|}. \tag{17}$$

It may be shown that this iterative scheme converges to $\bar{\psi}$ provided that $\alpha^p \to 0 \quad (p \to \infty)$ and $\sum_{p=0}^{\infty} \alpha^p = +\infty$. One can choose for instance $\alpha^p = 1/p$. The main drawback of this method is the fact that the convergence rate is usually very low.

5.1.2 The steepest descent method (1st order method)

In this frequently used method, α^p is chosen at each iteration p so as to minimize the function $g(\alpha) = j\left(\psi^p - \alpha \nabla j(\psi^p)\right)$ on the set of $\alpha \geq 0$. The algorithm is thus the following. One first chooses a starting point ψ^0 and set $p = 0$. At each iteration p, one computes the gradient and set $d^p = -\nabla j(\psi^p)$. One then solves the one-dimensional problem (see section 4) and set $\psi^{p+1} = \psi^p + \alpha^p d^p$. This procedure is repeated until a stopping test is satisfied (see section 2.5). The main disadvantage of the steepest descent method is the fact that the convergence can still be very slow. As a matter of fact, since α^p minimizes $g(\alpha) = j\left(\psi^p + \alpha d^p\right)$ then $g'(\alpha^p) = \left(d^p, \nabla j\left(\psi^p + \alpha d^p\right)\right) = \left(d^p, \nabla j\left(\psi^{p+1}\right)\right)$. Hence $\left(d^p, d^{p+1}\right) = 0$. This means that two successive displacements are strictly orthogonal. As a direct consequence, the number of steps to minimize elongated valley-type functions for instance may be very high (see figure 8 and then figure 10d page 23).

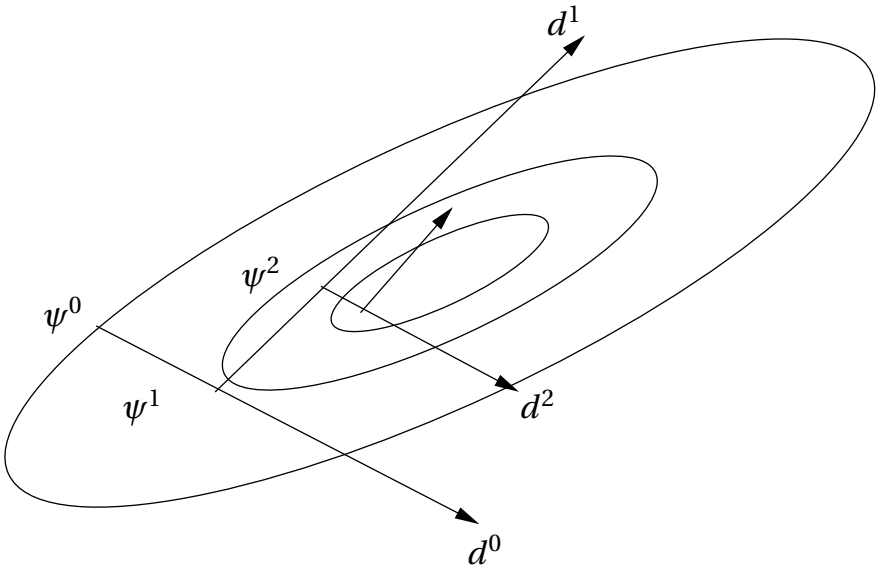


Figure 8: When the steepest descent method is used, the two consecutive directions are orthogonal.

5.1.3 The conjugate gradient method for quadratic functions (1st order method)

In this section we shall firstly assume that the cost function is quadratic. The case of arbitrary functions shall dealt with in the section 5.1.4 . Let the quadratic functional be of the form:

$$j(\psi) = \frac{1}{2}\left(\mathscr{A}\psi, \psi\right), \tag{18}$$

and let us recall the definition for two conjugate vectors. Let $\mathscr{A}$ be a given symmetric matrix (operator). Two vectors x_1 and x_2 are $\mathscr{A}$-conjugate if $(\mathscr{A}x_1, x_2) = 0$. The general method to optimize j is the following. Let us start with a given ψ^0 and choose $d^0 = -\nabla j(\psi^0)$. One may remark that for quadratic functions, the one-dimensional minimization procedure may be analytically solved. Recalling that the minimization of $g(\alpha)$ along the direction d^0 should lead to the fact that this current direction (d^0) would be orthogonal to the next gradient $\nabla j(\psi^1)$, one has:

$$\left(d^0, \nabla j\left(\psi^1\right)\right) = 0. \tag{19}$$

Using the relationship $\nabla j(\psi) = \mathscr{A}\psi$ given by the differentiation of (18) and the reactualization formulation $\psi^1 = \psi^0 + \alpha^0 d^0$, (19) becomes:

$$\begin{aligned} \left(d^0, \nabla j\left(\psi^1\right)\right) &= \left(d^0, \mathscr{A}\psi^1\right) \\ &= \left(d^0, \mathscr{A}\left(\psi^0 + \alpha^0 d^0\right)\right) \\ &= \left(d^0, \mathscr{A}\psi^0\right) + \alpha^0\left(d^0, \mathscr{A}d^0\right). \end{aligned} \tag{20}$$

Equaling (20) to zero gives the step size α^0:

$$\alpha^0 = -\frac{\left(d^0, \mathscr{A}\psi^0\right)}{\left(d^0, \mathscr{A}d^0\right)}. \tag{21}$$

Next, at stage p, we are at the point ψ^p and we compute the gradient $\nabla j(\psi^p)$. The direction d^p is obtained by combining linearly the gradient $\nabla j(\psi^p)$ and the previous direction d^{p-1}, where the coefficient β^p is chosen in such a way that d^p is $\mathscr{A}$-conjugate to the previous direction. Hence:

$$\begin{aligned} \left(d^p, \mathscr{A}d^{p-1}\right) &= \left(-\nabla j\left(\psi^p\right) + \beta^p d^{p-1}, \mathscr{A}d^{p-1}\right) \\ &= -\left(\nabla j\left(\psi^p\right), \mathscr{A}d^{p-1}\right) + \beta^p\left(d^{p-1}, \mathscr{A}d^{p-1}\right). \end{aligned} \tag{22}$$

Next, choosing β^p such that the previous equation equals zero yields to:

$$\beta^p = \frac{\left(\nabla j(\psi^p), \mathscr{A}d^{p-1}\right)}{\left(d^{p-1}, \mathscr{A}d^{p-1}\right)}. \tag{23}$$

The algorithm based on above relationships is te one given in Algorithm 2. Also, it is proved [3] that the conjugate gradient method applied on quadratic functions converges in at most n iterations where $n = \dim\psi$.

5.1.4 The conjugate gradient method for arbitrary (non quadratic) functions (1st order)

Before giving the conjugate gradient method to be applied on arbitrary functions, let us give some more properties inherent to quadratic functions. Differentiating (18) and taking into account of the reactualization relationship gives:

$$\begin{aligned} \nabla j(\psi^p) - \nabla j(\psi^{p-1}) &= \mathscr{A}\left(\psi^p - \psi^{p-1}\right) \\ &= \mathscr{A}\left(\psi^{p-1} + \alpha^{p-1}d^{p-1} - \psi^{p-1}\right) \\ &= \alpha^{p-1}\mathscr{A}d^{p-1}, \end{aligned} \tag{24}$$

Algorithm 2: The conjugate gradient algorithm applied on quadratic functions

1. Let $p = 0$, ψ^0 be the starting point,
   compute the gradient and the descent direction, $d^0 = -\nabla j(\psi^0)$,
   compute the step size $\alpha^0 = -\dfrac{\left(d^0, \mathscr{A}\psi^0\right)}{\left(d^0, \mathscr{A}d^0\right)}$;

2. At step p, we are at the point ψ^p.
   We define $\psi^{p+1} = \psi^p + \alpha^p d^p$ with:

   - the step size $\alpha^p = -\dfrac{\left(d^p, \nabla j(\psi^p)\right)}{(d^p, \mathscr{A}d^p)}$
   - the direction $d^p = -\nabla j(\psi^p) + \beta^p d^{p-1}$
   - where the coefficient needed for conjugate directions: $\beta^p = \dfrac{\left(\nabla j(\psi^p), \mathscr{A}d^{p-1}\right)}{\left(d^{p-1}, \mathscr{A}d^{p-1}\right)}$;

3. Stopping rule (see Section 2.5). If satisfies: End, otherwise set $p \leftarrow p+1$ and return to step (2).

which also gives the following relationship:

$$\frac{1}{\alpha^{p-1}}\left(\nabla j(\psi^p), \nabla j(\psi^p) - \nabla j(\psi^{p-1}\right) = \left(\nabla j(\psi^p), \mathscr{A}d^{p-1}\right). \tag{25}$$

On the other hand, substituting (25) into (23) gives

$$\beta^p = \frac{\left(\nabla j(\psi^p), \mathscr{A}d^{p-1}\right)}{\left(d^{p-1}, \mathscr{A}d^{p-1}\right)} = \frac{\left(\nabla j(\psi^p), \nabla j(\psi^p) - \nabla j(\psi^{p-1})\right)}{\left(d^{p-1}, \nabla j(\psi^p) - \nabla j(\psi^p - 1)\right)}. \tag{26}$$

Next, expanding the descent direction d^{p-1}, (26) becomes:

$$\beta^p = \frac{\left(\nabla j(\psi^p), \nabla j(\psi^p) - \nabla j(\psi^{p-1})\right)}{\left(-\nabla j(\psi^{p-1}) + \beta^{p-1}d^{p-2}, \nabla j(\psi^p) - \nabla j(\psi^{p-1})\right)}; \tag{27}$$

$$\beta^p = \frac{\left(\nabla j(\psi^p), \nabla j(\psi^p) - \nabla j(\psi^{p-1})\right)}{\left(-\nabla j(\psi^{p-1}) - \beta^{p-1}\nabla j(\psi^{p-2}) + \Lambda, \nabla j(\psi^p) - \nabla j(\psi^{p-1})\right)}, \tag{28}$$

where Λ is the series given from the reactualizations. All the gradients being orthogonal one from the next, (28) becomes:

$$\beta^p = \frac{\left(\nabla j(\psi^p), \nabla j(\psi^p) - \nabla j(\psi^{p-1})\right)}{\left(\nabla j(\psi^{p-1}), \nabla j(\psi^{p-1})\right)}, \tag{29}$$

and also:

$$\beta^p = \frac{\left(\nabla j(\psi^p), \nabla j(\psi^p)\right)}{\left(\nabla j(\psi^{p-1}), \nabla j(\psi^{p-1})\right)}. \tag{30}$$

It is pointed out that in the neighborhood of the optimum, non-quadratic functions may be always approximated by quadratic functions. The Fletcher and Reeves' method consists in

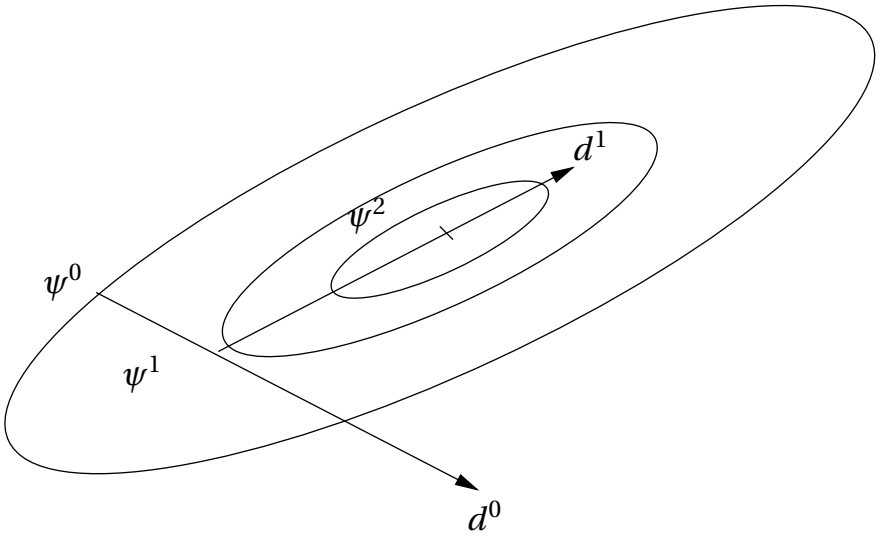


Figure 9: When the conjugate gradient method is used, the two consecutive directions are conjugate instead of orthogonal. Applied on a quadratic function, the method converges in at most n iterations (in this figure two iterations are needed since $\dim \psi = 2$).

applying (30) to access β^p while the Polak and Ribiere's method consists in applying (29) to access β^p. Taking into account of above remarks, the *conjugate gradient algorithm applied on arbitrary functions* is given in Algorithm 3. It is important to note that the global convergence of the presented methods is only ensured if a periodic restart is carried out . The restart $d^n = -\nabla j(u^n)$ is usually carried out at least every n iterations.

Algorithm 3: The conjugate gradient algorithm applied on arbitrary functions

1. Let $p = 0$, ψ^0 be the starting point, $d^0 = -\nabla j(\psi^0)$;
2. At step p, we are at the point ψ^p; we define $\psi^{p+1} = \psi^p + \alpha^p d^p$ with:
   - the step size $\alpha^p = \arg\min_{\alpha \in \mathbb{R}^+} g(\alpha) = j\left(\psi^p + \alpha d^p\right)$ with:
   - the direction $d^p = -\nabla j(\psi^p) + \beta^p d^{p-1}$ where
   - the conjugate condition β^p satisfies either (29) or (30) depending on the chosen method;
3. Stopping rule (see subsection 2.5). If satisfies: End, otherwise, set $p \leftarrow p+1$ and return to step (2).

5.2 The Newton's method (2nd order)

Let us assume that the cost function $j(\psi)$ is now twice continuously differentiable and that second derivatives exist. The idea is to approach the next cost function gradient by its quadratic

approximation through a Taylor development:

$$\nabla j(\psi^{p+1}) = \nabla j(\psi^p) + \left[\nabla^2 j(\psi^p)\right] \delta\psi^p + \mathscr{O}\left(\delta\psi^p\right)^2,$$

and equaling the obtained approximated gradient to zero to get the new parameter $\psi^{p+1} = \delta\psi^p + \psi^p$:

$$\psi^{p+1} = \psi^p - \left[\nabla^2 j\left(\psi^p\right)\right]^{-1} \nabla j(\psi^p). \tag{31}$$

Note that while using second-order optimization algorithms, the direction of descent as well as the step size are obtained from (31) in one go. Another interesting point is the fact that the algorithm converges to $\bar{\psi}$ in a single step when applied to strictly quadratic functions. However, for arbitrary functions, the approximation (**??**) may not be exact yielding to some errors in the displacement $\delta\psi^p$ and thus in the new point ψ^{p+1}. As a consequence, if the starting point ψ^0 is too far away from the solution $\bar{\psi}$, then the Newton method may not converge. On the other hand, since the approximation of $j(\psi)$ by a quadratic function is almost always valid in the neighborhood of $\bar{\psi}$, then the algorithm should converge to $\bar{\psi}$ if the starting point ψ^0 is chosen closely enough to the solution. Moreover, it is very common to control the step size this way. One first calculates the direction $d^p = -\left[\nabla^2 j(\psi^p)\right]^{-1} \nabla j(\psi^p)$ and control the step size through an iterative one-dimensional minimization problem of the kind $\min g(\alpha) = j\left(\psi^p + \alpha d^p\right)$ before actualization $\psi^{p+1} = \psi^p + \alpha d^p$. One limitation of the Newton's method is when the Hessian $\nabla^2 j(u^p)$ is not positive definite. In these cases, the direction given by $d^p = -\left[\nabla^2 j(\psi^p)\right]^{-1} \nabla j(u^p)$ may not be a descent direction, and the global convergence of the algorithm may not be guaranteed any more. Moreover, and above all, the Hessian is usually very difficult to compute and highly time consuming. To overcome these difficulties, one should prefer using one of the numerous quasi-Newton methods detailed afterwards.

5.3 The quasi-Newton methods

The quasi-Newton methods consist in generalizing the Newton's recurrence formulation (31). Since the limitation of the Newton's method is the restriction of the Hessian $\nabla^2 j(u^p)$ to be positive definite, the natural extension consists in replacing the *inverse of the Hessian* by an approximation to a positive definite matrix denoted $\mathbf{H}^p$. Obviously, this matrix is modified at each step p. There is much flexibility in the choice for computing the matrix $\mathbf{H}^p$. In general, the condition given by (32) is imposed:

$$\mathbf{H}\left[\nabla j(\psi^p) - \nabla j(\psi^{p-1})\right] = \psi^p - \psi^{p-1}. \tag{32}$$

Various corrections of the type

$$\mathbf{H}^{p+1} = \mathbf{H}^p + \Lambda^p \tag{33}$$

may be found in literature [2]. Depending on whether Δ^p is of rank 1 or 2, we shall speak of a correction of rank 1 or 2.

5.3.1 Rank 1 correction

The point is to choose a symetric matrix $\mathbf{H}^0$ and perform the corrections so that they preserve the symetry of the matrices $\mathbf{H}^p$. The rank 1 correction matrix consists in choosing $\Delta^p = \alpha^p v^p v^{p\top}$

where v^p is a vector and α^p is a scalar such that, from a symetric matrix $\mathbf{H}^0$, the correction preserves the symetry of matrices $\mathbf{H}^p$. Denoting

$$\delta^p = \psi^{p+1} - \psi^p \tag{34}$$

$$\gamma^p = \nabla j(\psi^{p+1}) - \nabla j(\psi^p) \tag{35}$$

one chooses α^p and v^p such that $\mathbf{H}^{p+1}\gamma^p = \delta^p$, thus:

$$\left[\mathbf{H}^p + \alpha^p (v^p v^{p\,t})\right] \gamma^p = \delta^p, \tag{36}$$

and

$$\gamma^{p\top} \mathbf{H}^p \gamma^p + \alpha^p \left(\gamma^{p\top} v^p\right)\left(v^{p\top} \gamma^p\right) = \gamma^{p\top} \delta^p, \tag{37}$$

thus

$$\alpha^p \left(v^{p\top} \gamma^p\right)^2 = \gamma^{p\top} \left(\delta^p - \mathbf{H}^p \gamma^p\right). \tag{38}$$

Using the identity

$$\alpha^p \left(v^p v^{p\top}\right) = \frac{\left(\alpha^p v^p v^{p\top} \gamma^p\right)\left(\alpha^p v^p v^{p\top} \gamma^p\right)^\top}{\alpha^p \left(v^{p\top} \gamma^p\right)^2}, \tag{39}$$

and using (36) and (37) to get

$$\alpha^p v^p v^{p\top} \gamma^p = \delta^p - \mathbf{H}^p \gamma^p, \tag{40}$$

$$\alpha^p \left(v^{p\top} \gamma^p\right)^2 = \gamma^{p\top} \left(\delta^p - \mathbf{H}^p \gamma^p\right), \tag{41}$$

one obtains the correction (of rank 1) of the Hessian inverse:

$$\mathbf{H}^{p+1} - \mathbf{H}^p = \alpha^p \left(v^p v^{p\top}\right) = \frac{\left(\delta^p - \mathbf{H}^p \gamma^p\right)\left(\delta^p - \mathbf{H}^p \gamma^p\right)^\top}{\gamma^{p\top}\left(\delta^p - \mathbf{H}^p \gamma^p\right)}. \tag{42}$$

5.3.2 The rank 2 Davidon-Fletcher-Powell (DFP) algorithm

The Davidon-Fletcher-Powell algorithm (in short DFP) consists in modifying the inverse hessian with the correction formulation of rank 2:

$$\mathbf{H}^{p+1} = \mathbf{H}^p + \frac{\delta^p (\delta^p)^\top}{(\delta^p)^\top \gamma^p} - \frac{\mathbf{H}^p \gamma^p (\gamma^p)^\top \mathbf{H}^p}{(\gamma^p)^\top \mathbf{H} \gamma^p} \tag{43}$$

where we have defined above $\delta^p = \psi^{p+1} - \psi^p$ and $\gamma^p = \nabla j(\psi^{p+1}) - \nabla j(\psi^p)$, and where the new point ψ^{p+1} is obtained from ψ^p through the displacement

$$d^p = -\mathbf{H}^p \nabla j(\psi^p). \tag{44}$$

The global DFP method is presented in Algorithm 4.

Algorithm 4: The Davidon – Fletcher – Powell (DFP) algorithm

1. Let $p = 0$, ψ^0 be the starting point. Choose any positive definite matrix $\mathbf{H}^0$ (often the identity matrix);
2. at step p, compute the displacement direction $d^p = -\mathbf{H}^p \nabla j(\psi^p)$, and find ψ^{p+1} at the minimum of $j(\psi^p + \alpha d^p)$ with $\alpha \geq 0$;
3. set $\delta^p = \psi^{p+1} - \psi^p$ and compute $\gamma^p = \nabla j(\psi^{p+1}) - \nabla j(\psi^p)$ to actualize:
$$\mathbf{H}^{p+1} = \mathbf{H}^p + \frac{\delta^p (\delta^p)^t}{(\delta^p)^t \gamma^p} - \frac{\mathbf{H}^p \gamma^p (\gamma^p)^t \mathbf{H}^p}{(\gamma^p)^t \mathbf{H} \gamma^p}; \tag{45}$$
4. Stopping rule (see section 3.4). If satisfies: End, otherwise, set $p \leftarrow p+1$ and return to step (2)

5.3.3 The rank 2 Broyden – Fletcher – Goldfarb – Shanno (BFGS) algorithm

The Broyden – Fletcher – Goldfarb – Shanno algorithm (in short BFGS) developped in 1969-1970 uses a rank 2 correction matrix for the inverse Hessian that is derived from (43). It can be shown [2] that the vectors δ^p and γ^p can permute in (43) and in the relationship $\mathbf{H}^{p+1}\gamma^p = \delta^p$. The correction (43) can thus also approximate the Hessian itself, and the correction for the inverse Hessian $\mathbf{H}^{p+1}$ can thus be given from $\mathbf{H}^p$ through the correction formulation:

$$\mathbf{H}^{p+1} = \mathbf{H}^p + \left[1 + \frac{\gamma^{p\,t} \mathbf{H}^p \gamma^p}{\delta^{p\,t} \gamma^p}\right] \frac{\delta^p (\delta^p)^t}{(\delta^p)^t \gamma^p} - \frac{\delta^p \gamma^{p\,t} \mathbf{H}^p + \mathbf{H}^p \gamma^p \delta^{p\,t}}{\delta^{p\,t} \gamma^p}. \tag{46}$$

When applied on a non purely quadratic function, one has, as for the conjugate gradient method and the DFP method, to carry out a periodic restart in order to ensure convergence [5, 12]. It is known that the BFGS algorithm is superior than the DFP algorithm is the sense that it is much less sensitive on the line-search inaccuracy allowing the use of economical inexact line-search algorithms [2].

5.3.4 Gauss–Newton

When the cost function is explicitly a square norm of the error between the prediction and the state, that is of the form

$$j(\psi) := \mathscr{J}(u) = \|u - u_d\|^2_{\mathscr{X}} \tag{48}$$

then the Gauss–Newton method or some derivatives or it (e.g. Levenberg–Marquardt) may be interesting to deal with, especially if the number of parameters is small. Before going deeper into the cost function gradient computation (see section 6), defining $u'(\psi;\delta\psi)$ the derivative of the state at the point ψ in the direction $\delta\psi$ as:

$$u'(\psi;\delta\psi) := \lim_{\epsilon \to 0} \frac{u(\psi + \epsilon\delta\psi) - u(\psi)}{\epsilon}, \tag{49}$$

then the directional derivative of the cost function writes:

$$j'(\psi;\delta\psi) = \left(u - u_d, u'(\psi;\delta\psi)\right)_{\mathscr{X}}, \tag{50}$$

Algorithm 5: The BFGS algorithm

1. Let $p = 0$, ψ^0 be the starting point. Choose any positive definite matrix $\mathbf{H}^0$ (often the identity matrix);
2. at step p, compte the displacement direction $d^p = -\mathbf{H}^p \nabla j(\psi^p)$, and find ψ^{p+1} at the minimum of $j(\psi^p + \alpha d^p)$ with $\alpha \geq 0$;
3. set $\delta^p = \psi^{p+1} - \psi^p$ and compute $\gamma^p = \nabla j(\psi^{p+1}) - \nabla j(\psi^p)$ to actualize:

$$\mathbf{H}^{p+1} = \mathbf{H}^p + \left[1 + \frac{\gamma^{p\,t}\mathbf{H}^p\gamma^p}{\delta^{p\,t}\gamma^p}\right]\frac{\delta^p(\delta^p)^t}{(\delta^p)^t\gamma^p} - \frac{\delta^p\gamma^{p\,t}\mathbf{H}^p + \mathbf{H}^p\gamma^p\delta^{p\,t}}{\delta^{p\,t}\gamma^p} \tag{47}$$

4. Stopping rule (see section 3.4). If satisfies: End, otherwise, set $p \leftarrow p+1$ and return to step (2)

where $j'(\psi;\delta\psi) = \big(\nabla j(\psi), \delta\psi\big)$. In the analogue way, the second derivative of $j(\psi)$ at the point ψ in the directions $\delta\psi$ and $\delta\phi$ is given by:

$$j''(\psi;\delta\psi,\delta\phi) = \big(u - u_d, u''(\psi;\delta\psi,\delta\phi)\big)_{\mathscr{X}} + \big(u'(\psi;\delta\psi), u'(\psi;\delta\phi)\big)_{\mathscr{X}}. \tag{51}$$

Neglecting the second-order term (this is actually the Gauss–Newton approach), we have:

$$j''(\psi;\delta\psi,\delta\phi) \approx \big(u'(\psi;\delta\psi), u'(\psi;\delta\phi)\big)_{\mathscr{X}}. \tag{52}$$

In order to form the cost function gradient vector and the approximated Hessian matrix, one has to choose the directions for the whole canonical base of ψ. Doing so, one can use the so-called sensititivity matrix S which gathers the derivatives of u in all directions $\delta\psi_i$, $i = 1,\ldots,\dim\psi$, and the product $\big(u'(\psi;\delta\psi_i), u'(\psi;\delta\psi_j)\big)$ involved in (52) is the product of the so-called sensitivity matrix with its transposed. The Newton relationship is thus approximated to:

$$S^t S\delta\psi^k = -\nabla j(\psi^k). \tag{53}$$

The matrix system $S^t S$ is obviously symmetric and positive definite with a dominant diagonal yielding thus to interesting features (Cholesky factorization, etc.). Though the Gauss–Newton system (53) presents inherent interesting feature (it almost gives in one step the descent direction and the step size), the matrix $S^t S$ is likely to be ill-conditionned. One way to decrease significantly the ill-condition feature is to "damp" the system using:

$$\big[S^t S + \ell I\big]\,\delta\psi^k = -\nabla j(\psi^k), \tag{54}$$

or better:

$$\big[S^t S + \ell\,\mathrm{diag}(S^t S)\big]\,\delta\psi^k = -\nabla j(\psi^k). \tag{55}$$

Note that $\ell \to 0$ yields the Gauss–Newton algorithm while ℓ bigger gives an approximation of the steepest descent gradient algorithm. In practice, the parameter ℓ may be adjusted at each iteration.

5.4 Elements of comparison between some presented methods

Some of the presented methods are tested using on the well-known Rosenbrock function which, in 2–D, writes:

$$f(x,y) = (x-\alpha)^2 + \beta(x^2-y)^2. \tag{56}$$

For the considered case, the chosen parameters are $\alpha = 10$ and $\beta = 100$, so that the optimum is at $(1,1)$. The figure 10a page 23 presents the function. This function presents a long elongated valley where the function gradient is very low. Next, the PSO algorithm is the one from [11].

The deterministic simplex method from the GSL library starting from the point $x^0 = -1$, $y^0 = 1$ needs 64 evaluations of the cost function. The stopping criterion is based on the simplex characteristic size equal to 10^{-2}. The PSO algorithm taken from [11] with 20 particles with 3 informed particles, $\phi = 4.14$, $\chi = \frac{2}{\phi-2+\sqrt{\phi^2-4\phi}}$, $\lambda_1 = \lambda_2 = 0.5\chi\phi$. The stopping criterion is based on the cost function equal to 10^{-5}. With these parameters, around 6,000 evaluations of the cost function is needed for the minimization. For the Steepest descent, the conjugate gradient and the BFGS algorithms, the stopping criterion is based either on the gradient norm equal to 10^{-3}, or on a maximum number of iterations equal to 10,000. For the steepest descent method, the maximum of iteration criterion is achieved. For the conjugate gradient, and the BFGS method, 49 and 11 iterations are needed, respectively.

6 Cost function gradient

We recall here that the function to be minimized is the cost function $\mathscr{J}(u)$ expressed in terms of the state u but minimized with respect to the parameters ψ. We thus have the equality (by definition) between the cost function and its reduced version: $j(\psi) := \mathscr{J}(u)$. The state u is related to the parameters ψ through an operator (linear or not) that combines the partial differential equations along with the boundary conditions, initial conditions, etc. This operator is denoted as $\mathscr{S}$ for the state problem. To be concise, one writes down

$$\mathscr{S}(u,\psi) = 0, \tag{57}$$

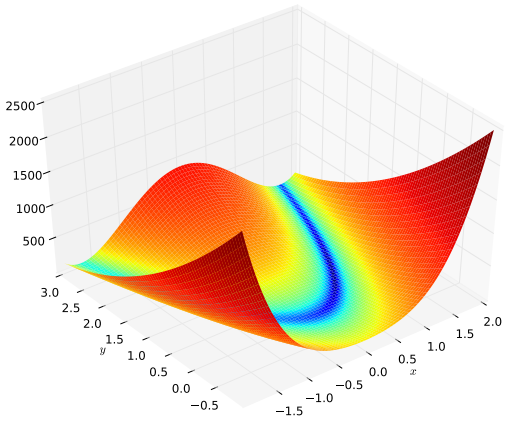
where we have the mapping $\psi \mapsto u(\psi)$. Often, the space (and time) is discretized so that the state operator $\mathscr{S}$ is approximated in some matrix formulation. In this case, we have $\mathscr{R}(u,\psi) = 0$, with $\dim\mathscr{R} = \dim u$. Note that u involved in (57) is continuous while u involved in $\mathscr{R}(u,\psi) = 0$ is likely to be already discretized (using finite difference, finite elements, etc.). We now need the definition of the directional derivative of $j(\psi)$ in the direction $\delta\psi$ (see Definition 6). Other kinds of derivatives can also be used, such as the Gâteaux or Fréchet derivatives, see [1] for technical definitions.

Definition 6 (Directional derivative)**.** *Let a point $\psi \in \mathscr{K}$ and a direction $\phi \in \mathscr{K}$. One defines $\ell(t) := \psi + t\phi$ and the function $\mathscr{J}(t) := j(\ell(t))$. The directional derivative of j at the point ψ in the direction ϕ is:*

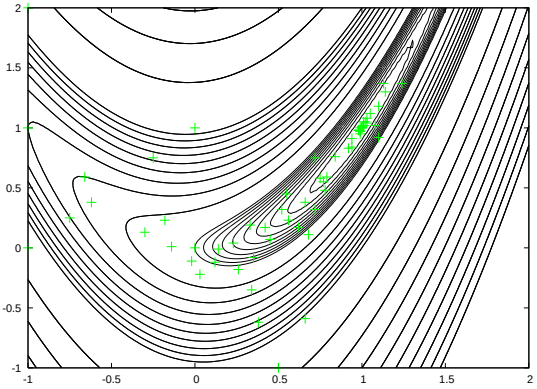
$$j'(\psi;\phi) := \mathscr{J}'(0) = \lim_{\substack{t\to 0 \\ t>0}} \frac{j(\psi+t\phi) - j(\psi)}{t}. \tag{58}$$

It has been seen before (see (50)) that we have the equality

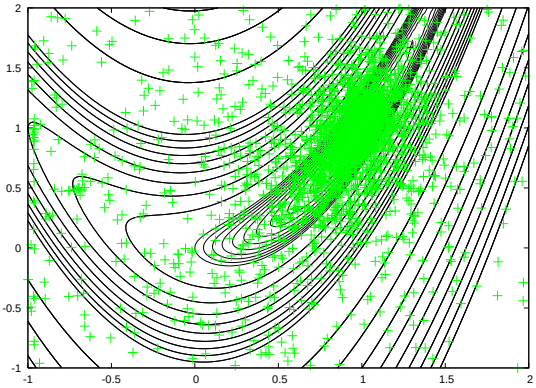
$$j'(\psi;\delta\psi) = \big(u-u_d, u'(\psi;\delta\psi)\big)_{\mathscr{X}}, \tag{59}$$



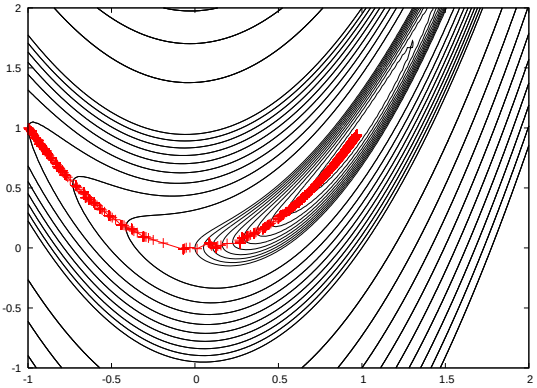
(a) 2–D Rosenbrock function.



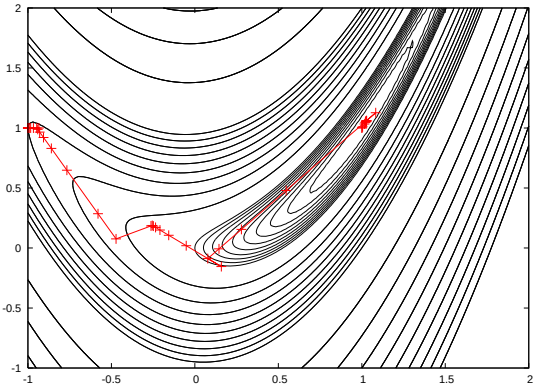
(b) Simplex algorithm: 64 cost function evaluations.



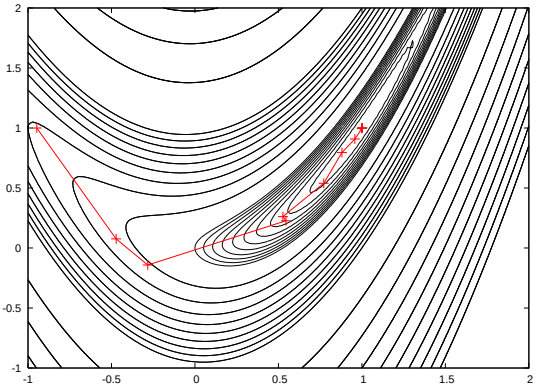
(c) PSO algorithm: ≈ 6,000 cost function evaluations.



(d) Steepest descent algorithm: more than 10,000 cost function evaluations.



(e) Conjugate gradients descent algorithm: 45 cost function evaluations.



(f) BFGS descent algorithm: 11 cost function evaluations.

Figure 10: Numerical comparison of optimizers on the 2-D Rosenbrock function.

and, because of linearity of both $u'(\psi;\delta\psi)$ and $j'(\psi;\delta\psi)$ in $\delta\psi$:

$$j'(\psi;\delta\psi) = \big(\nabla j(\psi), \delta\psi\big)_{\mathcal{Z}}. \tag{60}$$

where $\mathcal{Z}$ is most of the time chosen equal to $\mathcal{Y}$ but it can be chosen differently for regularization purposes.

6.1 Finite difference

The finite difference approach consists in approaching the cost function gradient through a substraction of the cost function with a perturbed cost function for the whole canonical base of ψ, that is $\delta\psi = \delta\psi_1, \delta\psi_2, \ldots, \delta\psi_{\dim\psi}$. For the i^{th} component, we have:

$$\big(\nabla j(\psi)\big)_i = \big(\nabla j(\psi), \delta\psi_i\big)_{\mathcal{Z}} \approx \frac{j(\psi+\epsilon\delta\psi_i) - j(\psi)}{\epsilon}. \tag{61}$$

Usually, in order to perform the same relative perturbation on all components ψ_i, one rather uses $\epsilon_i \leftarrow \varepsilon|\psi_i|$, where the positive scalar ε is fixed. The very simple related algorithm is described in Algorithm 6.

Algorithm 6: The finite difference algorithm to compute the gradient of the cost function

Set the length $\varepsilon > 0$;
At iteration p, compute the state $u(\psi^p)$, compute $j(\psi^p)$;
foreach $i = 1, \ldots, \dim\psi$ **do**
  Compute the cost $j(\psi^p + \varepsilon|\psi_i|\delta\psi_i)$;
  Set the gradient $(\nabla j(\psi))_i \leftarrow \dfrac{j(\psi^p + \varepsilon|\psi_i|\delta\psi_i) - j(\psi^p)}{\varepsilon|\psi_i|}$
end
Integrate the gradient within the optimization methods that do not rely on the sensitivities (conjugate gradient or BFGS for instance among the presented methods)

In practice the tuning parameter ε has to be chosen within a region where variables depend roughly linearly on ε. Indeed for too small values, the round-off errors dominate while for too high values one gets a nonlinear behavior. Even though the Finite Difference Method is easy to implement, it has the disadvantage of being highly CPU time consuming. Indeed, the method needs as many integrations of the model $\mathcal{S}(u,\psi) = 0$ as the number of parameters $\dim\psi$. The gradient computed this way can be integrated to the previously presented optimization methods that do not rely on u', such as the conjugate gradient methods or better the BFGS.

When performing the finite differenciation with respect to ψ_i, one also accesses the approximated perturbed state $u'(\psi;\delta\psi_i)$. This way, one can use again the conjugate gradient methods or the BFGS method for instance, but also the Gauss–Newton-type methods based on matrix inversion and which do rely on the sensitivities $u'(\psi;\delta\psi_i)$, $i = 1, \ldots, \dim\psi$. Doing so, the related optimization is given in Algorithm 7.

6.2 Forward differentiation

The forward differentiation approach consists in computing $u'(\psi;\delta\psi_i)$ differentiating the state equations $\mathcal{S}(u,\psi) = 0$ to get:.

$$\mathcal{S}'_u(u,\psi)\,u' + \mathcal{S}'_\psi(u,\psi)\delta\psi = 0. \tag{62}$$

Algorithm 7: The finite difference algorithm to compute the gradient of the cost function and the sensitivities

Set the step $\varepsilon > 0$;
At iteration p, compute the state $u(\psi^p)$, compute $j(\psi^p)$;
foreach $i = 1, \dots, \dim \psi$ **do**
  Compute the perturbed state $u(\psi^p + \varepsilon|\psi_i|\delta\psi_i)$ and the cost $j(\psi^p + \varepsilon|\psi_i|\delta\psi_i)$;
  Set the state sensitivity $u'(\psi;\delta\psi_i) \leftarrow \dfrac{u(\psi^p + \varepsilon|\psi_i|\delta\psi_i) - u(\psi^p)}{\varepsilon|\psi_i|}$;
  Set the gradient $(\nabla j, \delta\psi_i)$ with either $\big(u - u_d, u'(\psi;\delta\psi_i)\big)$ or as in previous algorithm with $\dfrac{j(\psi^p + \varepsilon|\psi_i|\delta\psi_i) - j(\psi^p)}{\varepsilon|\psi_i|}$.
end
Integrate the gradient within the optimization methods that do not rely on the sensitivities (conjugate gradient or BFGS among the presented methods) or within optimization methods that do rely on the sensitivities (Gauss–Newton or Levenberg–Marquardt).

As in the previous section, the gradient computation needs one integration of (62) per parameter ψ_i, so one needs $\dim\psi$ integrations in total to access the full gradient $\nabla j(\psi)$. However, in this case, the equation (62) is linear, while (57) was not linear.

As for the finite difference approach, one may use the sensitivities u' and integrate them into the Gauss–newton-type methods, or simply use the cost function gradient and then use the methods that do not rely on the sensitivities.

When compared to the finite difference approach, the forward difference method leads to exact cost function gradient components. Moreover, though $\mathscr{S}$ is likely to be a nonlinear operator, the system (62) is linear, thus yielding to much less CPU time. Another singularity is that the discrete version of $\mathscr{S}'_u(u,\psi)$, i.e. $\mathscr{R}'_u$ is the tangent matrix that is to be used anyway for solving the "forward" problem $\mathscr{S}(u,\psi) = 0$. The computation of this linear tangent matrix is most often the task that takes the longer time in solving $\mathscr{S}(u,\psi) = 0$. The optimized procedure is thus the one given in Algorithm 8.

6.3 Adjoint state

In this section we present the use of an additional problem –the so-called adjoint-state problem– that gives the exact cost function gradient but in a computational cheap way. We present one method based on the identification procedure (subsection 6.3.1) and another one that uses the Lagrange function (subsection 6.3.2). For the latter method, the model equation is treated as an equality constraint for the optimization. Both methods can deal with either the continuous equations or the discrete ones. One has to keep in mind that when the continuous method is used, (in general) all the equations have later on to be discretized. Both strategies are equivalent in usual, but if the cost is computed through the integration of some discretized equations, then we consider that the discretized equations have to be differented (it is the so-called "discretize-then-differentiate" method). The other way is to deal with the continuous equations, then discretize the state model, etc. (it is the so-called "differentiate-then-discretize" method). Some examples of adjoint derivation will be given in the last sections.

Algorithm 8: The forward differentiation algorithm to compute the cost gradient and the sensitivities

At iteration p, solve iteratively $\mathscr{S}(u,\psi^p)=0$, compute $j(\psi^p)$ and save the discrete version of the linear tangent operator $\mathscr{S}'_u(u,\psi^p)$;
foreach $i=1,\dots,\dim\psi$ **do**
  Solve $\mathscr{S}'_u(u,\psi)u' + \mathscr{S}'_\psi(u,\psi^p)\delta\psi_i = 0$;
  Set $(\nabla j,\delta\psi_i)_{\mathcal{Z}} = \big(u-u_d, u'(\psi;\delta\psi_i)\big)_{\mathcal{X}}$;
end
Integrate the gradient within the optimization methods that do not rely on the sensitivities (conjugate gradient or BFGS among the presented methods) or within optimization methods that do rely on the sensitivities (Gauss–Newton or Levenberg–Marquardt).
Note: the linear tangent matrix which is to be assembled for the solution of the "forward" model can be re-used for all canonical components $\delta\psi_i$.

6.3.1 Identification method

In this first part, we derive the adjoint-state method using the identification method. From the definition of the functional gradient, one writes the gradient:

$$\big(\nabla j,\delta\psi\big)_{\mathcal{Z}} = j'(\psi;\delta\psi) = \big(u-u_d, u'(\psi;\delta\psi)\big)_{\mathcal{X}}. \tag{63}$$

One then introduces a new variable (the adjoint-state variable u^*) such that the gradient equation (63) also satisfies the "more–easy–to–compute":

$$j'(\psi;\delta\psi) = \Big(\mathscr{S}'_\psi(u,\psi)\delta\psi, u^*\Big)_{\mathcal{U}}. \tag{64}$$

On the other hand, since we have the relationship $\mathscr{S}(u,\psi)=0$, then

$$\mathscr{S}'_u(u,\psi)u' + \mathscr{S}'_\psi(u,\psi)\delta\psi = 0 \tag{65}$$

and thus

$$j'(\psi;\delta\psi) = -\big(S'_u(u,\psi)u', u^*\big)_{\mathcal{U}}. \tag{66}$$

Identifying (63) and (66), we obtain the adjoint-state problem that must satisfy the equality:

$$-\big(\mathscr{S}'_u(u,\psi)u', u^*\big)_{\mathcal{U}} = \big(u-u_d, u'(\psi;\delta\psi)\big)_{\mathcal{X}}. \tag{67}$$

Next, if the adjoint problem (67) is satisfied (it means that we accessed the adjoint state u^*), then the cost function gradient is very simply given by (64). We then use the inner product property $(\mathscr{A}u,v)=(u,\mathscr{A}^*v)$ where $\mathscr{A}^*$ is the transposed conjugate operator of $\mathscr{A}$ (adjoint) to modify the adjoint equation (67) to:

$$\mathscr{S}^*(u,\psi)u^* + (u-u_d) = 0 \tag{68}$$

where $\mathscr{S}^*$ is the conjugate transposed of the linear tangent operator $\mathscr{S}'_u$, i.e. we used:

$$\big(\mathscr{S}'_u(u,\psi)u', u^*\big)_{\mathcal{U}} = \big(S^*(u,\psi)u^*, u'\big)_{\mathcal{U}} + [\cdots] \tag{69}$$

where the term $[\cdots]$ may contain integration terms.

Remark. *The inner product $(v,w)_{\mathcal{U}}$ is performed on the whole domain of definition of u. For instance if $u\in L_2(0,T;L_2(\mathscr{D}))$, then $(v,w)_{\mathcal{U}} = \int_0^T\int_{\mathscr{D}} vw\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}t$.*

Algorithm 9: The adjoint state problem to compute the cost function gradient with integration within an optimization algorithm

At iteration p, solve iteratively $\mathscr{S}(u,\psi)=0$;
Compute $j(\psi^p)$;
Save the solution u;
Compute the adjoint state problem $\mathscr{S}^*(u,\psi)u^*+(u-u_d)=0$;
Compute the gradient $\left(\nabla j(\psi);\delta\psi\right)_{\mathscr{Z}}=\left(\mathscr{S}'_\psi(u,\psi)\delta\psi,u^*\right)_{\mathscr{U}}$;
Integrate the gradient within the optimization methods that do not rely on the sensitivities (conjugate gradient or BFGS among the presented methods)

6.3.2 Lagrange formulation

The use of a Lagrange formulation means that the state equations are taken as constraints in the optimization problem. To do so, let us introduce the Lagrange function [13, 14]:

$$\mathscr{L}(u,u^*,\psi)=\mathscr{J}(u)+\left(\mathscr{S}(u,\psi),u^*\right)_{\mathscr{U}} \tag{70}$$

The Lagrange function introduced in this section is a function of three variables, namely the state u , the parameter to be to be identified ψ and the adjoint state variable u^*. This means that both variables u and ψ are somehow considered to be independent even though there exists (at least implicitly) the relationship $\mathscr{S}(u,\psi)=0$ that maps ψ to u. Moreover, since u is the solution of the forward model, then the Lagrange function $\mathscr{L}$ is always equal to the cost function $\mathscr{J}(u)$ and the constraints which represent the partial differential equations of the modelling problem are always satisfied. We now show that a necessary condition for the set ψ to be solution of the optimization problem (2.3) is that there exists a set (u,ψ) such that (u,ψ,u^*) is a saddle point (stationary point) of $\mathscr{L}$. Indeed, let us show that the necessary condition $j'(\psi;\delta\psi)=0$, $\forall\delta\psi$ is equivalent to

$$\exists\left(u,u^*,\psi\right)\mid\mathscr{L}'_u(\cdot)\,\delta w=0;\ \mathscr{L}'_{u^*}(\cdot)\,\delta w=0;\ \mathscr{L}'_\psi(\cdot)\,\delta w=0, \tag{71}$$

for all directions δw taken in appropriate spaces (u', δu^* and $\delta\psi$). First, since the state is satisfied, then

$$\mathscr{L}'_{u^*}=\mathscr{S}(u,\psi)=0.$$

Moreover, since we have also $\mathscr{S}'_u(u,\psi)u'+\mathscr{S}'_\psi(u,\psi)\delta\psi=0$, we get:

$$\mathscr{L}'_\psi(\cdot)\,\delta\psi=\left(\mathscr{S}'_\psi(u,\psi)\delta\psi,u^*\right)_{\mathscr{U}}=-\left(\mathscr{S}'_u(u,\psi)u',u^*\right)_{\mathscr{U}} \tag{72}$$

In another hand, the differentiation of the Lagrange function with respect to the state gives:

$$\mathscr{L}'_u(\cdot)\,u'=\left(u-u_d,u'\right)_{\mathscr{X}}+\left(\mathscr{S}'_u(u,\psi)u',u^*\right)_{\mathscr{U}} \tag{73}$$

So far, the choice for the adjoint variables u^* has not been fixed yet. However, choosing the adjoint variable such that $\mathscr{L}'_u(\cdot)\,u'=0\ \forall u'$ considerably simplifies the relationship between the differentiated lagrangian with respect to ψ and the cost function gradient. One actually chooses u^* such that it satisfies the adjoint state equation

$$\left(S'_u(u,\psi)u',u^*\right)_{\mathscr{U}}+\left(u-u_d,u'(\psi;\delta\psi)\right)_{\mathscr{X}}=0. \tag{74}$$

This way we obtain the cost function gradient:

$$\mathscr{L}'_\psi(\cdot)\,\delta\psi = \big(u-u_d, u'(\psi;\delta\psi)\big)_{\mathscr{X}} = j'(\psi;\delta\psi) = \big(\nabla j, \delta\psi\big)_{\mathscr{Y}} \tag{75}$$

The adjoint-state equation is thus:

$$\mathscr{S}^*(u,\psi)\,u^* + (u-u_d) = 0, \tag{76}$$

and the gradient is given by:

$$\nabla j = \Big(\mathscr{S}'_\psi(u,\psi), u^*\Big)_{\mathscr{Y}}. \tag{77}$$

Summarizing, the minimum of the cost function is to be found at the stationary point of the Lagrange function (70). When the adjoint-state equation (76) is satisfied, then the components of the cost function gradient are simply given through the inner product (77).

6.3.3 Examples

In the examples presented below, we do not specify what the parameters are. We just give the form of the adjoint-state problem related to the "forward" state problem form.

Case of ODE Let us start with the case where the state model is simplified to a single linear continuous ordinary differential equations integrated in time $\mathscr{I} =]0, t_f]$. The direct problem thus writes:

$$\begin{array}{ll} \mathscr{S}\big(u,\psi\big) = \mathscr{C}\frac{\partial u}{\partial t} - \mathscr{B} = 0 & \text{for } t \in \mathscr{I} \\ u = u_0 & \text{for } t = 0, \end{array} \tag{78}$$

where $\mathscr{C}$ may be an inertial scalar or a capacity matrix depending on the related case and $\mathscr{B}$ contains the loadings. Injecting the differentiated time-dependent relationship (78) into the adjoint-state relationship (74) gives:

$$\left(\mathscr{C}\frac{d}{dt}u', u^*\right)_{\mathscr{U}} + \big(u-u_d, u'\big)_{\mathscr{X}} = 0$$

where the inner must be understood as $(a,b)_{\mathscr{U}} = \int_{\mathscr{I}} ab\,\mathrm{d}t$. One then integrates by part the first term to get:

$$-\left(u', \mathscr{C}^*\frac{d}{dt}u^*\right)_{\mathscr{U}} + \big[u'\mathscr{C}^*u^*\big]_0^{t_f} + \big(u-u_d, u'\big)_{\mathscr{X}} = 0$$

Since there is no reason that the initial state depend on the parameters ψ (except if the initial state is searched), then the derivatives u' of u at initial time is set to zero. The adjoint-state problem is eventually:

$$\begin{array}{ll} -\mathscr{C}^*\frac{\partial u^*}{\partial t} + (u-u_d) = 0 & \text{for } t \in \mathscr{I} \\ \mathscr{C}^*u^* + (u-u_d) = 0 & \text{for } t = t_f\,. \end{array} \tag{79}$$

Remark. *There is a minus sign just before the operator $\mathscr{C}^*$ involved in the first equation. At the same time, the boundary-time condition is given at final time t_f. Therefore, when considering these two points, there is no way to solve the adjoint problem forwardly, i.e. from $t=0$ to t_f. The trick consists in introducing a new time variable $\tau = t_f - t$ (the dual time). Doing so, the initial condition is given at initial time $\tau = 0$, and the time-dependent equation (79) is solved in the forward way in the dual time variable τ which corresponds to the backward way in the primal time variable t.*

Remark. *The loading component $(u - u_d)$ involved in (79) is non-zero only at times where the cost function j is to be integrated, i.e. in accordance with the definition of the $\mathscr{X}$-norm.*

Remark. *Inherently, the adjoint-state problem is linear: even though the forward problem is likely to be nonlinear (it was not the case in the considered exemple), the adjoint-state problem is still linear since the operators do not depend on the adjoint-state variables. An equivalent remark was given for the forward differentiation method which used the linear tangent operator.*

Case of elliptic PDE This second example concerns the case where the state model is simplified to a diffusive-type continuous partial differential equation independent of time:

$$\mathscr{S}\left(u,\psi\right) = \mathscr{K}\,u - \mathscr{B} = 0 \tag{80}$$

where $\mathscr{K}$ is the diffusivity matrix obtained after discretization of the continuous equation which takes into account of the boundary conditions. Injecting the differentiated space-dependent relation (80) into the adjoint (74) gives:

$$\left(\mathscr{K}\,u', u^*\right)_{\mathscr{U}} + \left(u - u_d, u'\right)_{\mathscr{X}} = 0$$

Transposing the diffusive matrix gives:

$$\left(u', \mathscr{K}^*\,u^*\right)_{\mathscr{U}} + \left(u - u_d, u'\right)_{\mathscr{X}} = 0$$

Owing to be verified for all directional derivatives u', the general adjoint-state problem becomes:

$$\mathscr{K}^*\,u^* + (u - u_d) = 0. \tag{81}$$

Remark. *The loading component involved in the space-dependent equation is non-zero only at the selected locations where the cost function j is to be integrated, i.e. in accordance with the definition of the $\mathscr{X}$-norm.*

Case of parabolic PDE The discretization of the space and time dependent diffusive model yields to the so-called parabolic problem. It is somehow the union between both just above presented cases:

$$\begin{array}{ll} \mathscr{S}\left(u,\psi\right) = \mathscr{C}\,\dot{u} + \mathscr{K}\,u - \mathscr{B} = 0 & \text{for } t \in \mathscr{I} \\ u = u_0 & \text{for } t = 0. \end{array} \tag{82}$$

Injecting the differentiated operators involved in (82) into the adjoint (74) gives:

$$\left(\mathscr{C}\frac{d}{dt}u', \psi\right)_{\mathscr{U}} + \left(\mathscr{K}\,u', \psi\right)_{\mathscr{U}} + \left(u - u_d, u'\right)_{\mathscr{X}} = 0$$

Transposing all operators through integration by parts gives:

$$-\left(u', \mathscr{C}^*\frac{d}{dt}u^*\right)_{\mathscr{U}} + \left[\left(u', \mathscr{C}^*\,u^*\right)_{\mathscr{D}}\right]_0^T + \left(u', \mathscr{K}^*\,u^*\right)_{\mathscr{U}} + \left(u - u_d, u'(\psi;\delta\psi)\right)_{\mathscr{X}} = 0$$

Eventually, the adjoint problem writes:

$$\begin{array}{ll} -\mathscr{C}^*\,\dot{u}^* + \mathscr{K}^*\,u^* + \mathscr{J}'(u) = 0 & \text{for } t \in \mathscr{I} \\ u^* = 0 & \text{for } t = t_f. \end{array} \tag{83}$$

6.3.4 The global optimization algorithm

The general algorithm is given in Algoritm 10. The global procedure described in this algorithm is run until (at least) one of the stopping criteria presented in subection 2.5 is reached.

Algorithm 10: The global optimization algorithm

1. Integrate the cost function value through integration of the forward (maybe nonlinear) problem;

   Store all state variables to reconstruct the tangent matrix (or store the tangent matrix);

2. Integrate the backward linear adjoint-state problem, all matrices being possibly stored or recomputed from stored state variables

3. Compute the cost function gradient;

   Compute the direction of descent

4. Solve the line-search algorithm

7 Elements of comparison

We give in this section some elements of comparison between the previously presented optimization algorithms and between the different gradient computation strategies.

7.1 Convergence speed

The optimization algorithms presented in section 5 yield to a series $\{\psi^k\}_{k\geq 1}$ that converges to $\bar{\psi}$. We give some rate convergence definitions [4, 2, 3].

Definition 7. *The convergence rate of the series $\{\psi^k\}_{k\geq 1}$ is said to be linear if*

$$\frac{\|\psi_{k+1}-\psi_k\|}{\|\psi_k-\bar{\psi}\|}\leq \tau, \quad \tau\in(0,1). \tag{84}$$

This means that the distance to the solution $\bar{\psi}$ decreases at each iteration by at least the constant factor τ.

Definition 8. *The convergence rate of the series $\{\psi^k\}_{k\geq 1}$ is said to be superlinear in n steps if*

$$\lim_{k\to\infty}\frac{\|\psi_{k+n}-\psi_k\|}{\|\psi_k-\bar{\psi}\|}=0. \tag{85}$$

Definition 9. *The convergence rate of the series $\{\psi^k\}_{k\geq 1}$ is said to be quadratic if*

$$\frac{\|\psi_{k+1}-\psi_k\|}{\|\psi_k-\bar{\psi}\|^2}\leq \tau, \quad \tau>0. \tag{86}$$

Quasi–Newton methods usually converge superlinearly and the Newton method converges quadratically. The steepest descent method converge linearly. Moreover, for ill-posed problems,

this method may converge linearly with a constant τ close to 1. Next, the conjugate-gradient method converges superlinearly in n steps to the optimum [2].

Thus the quasi-Newton methods convergence-rate is much higher than the conjugate gradient methods convergence rate which need approximatively n times more steps (n times more line-search) at the same convergence behavior [2]. However, for the quasi-Newton method, the memory place is proportionnal to n^2.

7.2 Gradient computation cost

Let $\mathscr{S}(u,\psi)=0$ the state problem that maps $\psi \mapsto u$, $\mathscr{R}$ being possibly nonlinear (for highlighting differences between the distinct strategies), and $\dim\psi$ the number of parameters to be evaluated. We compare the number of times the model $\mathscr{S}$, the differentiated model and/or the adjoint state model are computed to access the full gradient of the cost function.

1. Finite difference method:
   $(\dim\psi+1)$ nonlinear computation of $\mathscr{S}(u,\psi)=0$.

2. Forward differentiation method:
   1 nonlinear computation of $\mathscr{S}(u,\psi)=0$,
   $\dim\psi$ linear computation of $\mathscr{S}'_u(u,\psi)u'+\mathscr{S}'_\psi(u,\psi)\delta\psi=0$.

3. Adjoint state method:
   1 nonlinear computation of $\mathscr{S}(u,\psi)=0$,
   1 linear computation of $\mathscr{S}^*(u,\psi)u^*+u-u_d=0$.

Thus, the finite difference method is very time consuming, though it is easy to use. Next, comparing the two latter methods, the operator involved in the adjoint-state method is almost the same as the one involved in the forward differentiation method, though the adjoint-state method yields to higher algorithmic complexity (backward time integration, memory, etc.). When $\dim\psi$ is high (even if $\dim\psi$ is bigger than say 100), the use of the direct differentiation method becomes cumbersome and computationaly expensive: the adjoint-state is the only acceptable method.

7.3 Gradient computation needs

We recall in the following table the way (the required needed steps) one computes the cost function gradient.

Steepest, conjugate-gradients, BFGS, DFP, …	Newton	Gauss–Newton, Levenberg–Marquardt, …
$u \leftarrow \mathscr{S}(u,\psi)=0$ $j \leftarrow u$ $\nabla j \leftarrow \left\{ \begin{array}{l} \text{Forward diff.} \\ \text{or} \\ \text{Adjoint state} \end{array} \right.$	$u \leftarrow \mathscr{S}(u,\psi)=0$ $j \leftarrow u$ $\nabla j \leftarrow \left\{ \begin{array}{l} \text{Forward diff.} \\ \text{or} \\ \text{Adjoint state} \end{array} \right.$ $\nabla^2 j$ (complicated)	$u \leftarrow \mathscr{S}(u,\psi)=0$ $j \leftarrow u$ $\nabla j \leftarrow S^t S \leftarrow S \leftarrow u'$ (Forward diff.)

8 Regularization

When the inverse problem is ill-posed (which is likely to be the case in real cases, especially when the control space dimension is big), regularization is needed and sometimes compulsary. Regarding function estimation, for instance space-dependent physical properties or sources, specific regularization strategies different from the ones used in parametric estimation are required. Regularization may be viewed as adding *a priori* information, but other means can also be used, including (see [15, 16] for elements of comparison on applications of optical tomography):

- choose of specific $\mathscr{X}$-norm for the cost function expression according to the prior knowledge of the unknown (use for instance the $L_1(\mathscr{D})$-norm instead of the ordinary $L_2(\mathscr{D})$-norm).

- add prior information through Tikhonov penalization, If some Tikhonov-type regularization terms are added to the cost function, the cost function $j_\epsilon(\psi) := \mathscr{J}(u) + \epsilon \mathscr{J}^+(\psi)$ is the one to be minimized, with:
$$\mathscr{J}^+ := \left\| \mathbf{D}\psi \right\|^2_{\mathscr{Y}} \tag{87}$$
where $\mathbf{D}$ is often a differential operator acting on the function ψ and $\|\cdot\|_{\mathscr{Y}}$ is another norm to be defined according to the chosen control space.

- choose appropriate $\mathscr{Z}$-norm (or rather inner product) when extracting the cost function gradient, The use of specific inner products when extracting the cost function gradient is a recent regularization tool. In order to present this regularization strategy, let us work on the example where a space-dependent physical property in $\mathscr{D}$ is to be estimated. In such case it is usual to use the ordinary $L_2(\mathscr{D})$-inner product, i.e. one uses $j'(\psi;\phi) = \left(\nabla j, \phi\right)_{L_2(\mathscr{D})}$ can gives the ordinary $L_2(\mathscr{D})$ cost function gradient denoted as $\nabla^{L_2} j(\psi)$. But the Sobolev inner product can give much better (smoother) results when the noise has propagated to the adjoint-state variable and then to the cost function gradient. Even better, the weighted version has recently proven to give excellent results. This one defined as:
$$(u,v)_{\mathscr{Z}} = (u,v)_{H^{1(\ell)}(\mathscr{D})} := \int_{\mathscr{D}} \left(uv + \ell^2 \nabla u \cdot \nabla v \right) \, \mathrm{d}\boldsymbol{x} \tag{88}$$
is used in the cost function gradient extraction relationship $j'(\psi;\phi) = \left(\nabla j, \phi\right)_{H^{1(\ell)}(\mathscr{D})}$ in order to compute the weighted Sobolev cost function gradient $\nabla^{H^{1(\ell)}} j(\psi)$.

- choose appropriate functional space for the control space parameterization. In practice, in the control space must be approached in order to be finite. Often, the finite element method is used so that one searches ψ that belongs to a finite dimensional subspace, say $\mathscr{V}$. Let us consider a triangulation $\mathscr{M}$ of the computational domain $\mathscr{D}$, and let us note n_p the number of vertices in $\mathscr{M}$. It has been shown, through numerical means on a specific OT problem [15] that, among the large number of tested possibilities, the piecewise linear continuous functions ($\dim \psi = n_p$) are the most appropriate for the estimation of space-dependent functions.

- choose appropriate dimension $\dim \psi$ of the control space parameterization. Usually the finite element space used to solve the forward model (57) has to be fine enough to ensure

numerical errors small enough. Most often, the triangulation chosen for the control space is the one chosen for the state. It has been shown again, through numerical means, that both the convergence and the quality of the reconstructions are much improved when $\dim \psi$ is lowered, up to a certain limit, at least for quasi-Newton algorithms.

- Multi-scales approaches is also a fabulous opportunity to regularize solutions and in the same time accelerate convergence and avoid converging to local minima. Coupled with wavelets on one side, and the BFGS in the other side, this method relies on a reformulation of the original inverse problem into a sequence of sub-inverse problems of different scales using wavelet transform, from the largest scale to the smallest one. Successful applications of this method include the estimation of space-dependent obsorption and scattering coefficients in optical tomography [17].

9 Examples

9.1 Parametric conductivities in transient IHCP

This first example deals with the estimation of conductivity coefficients constant in different sub-domains. Heat transfer is considered. Initial temperature is assumed to be known and equal to T_0. T_0 is also the Dirichlet temperature for positive time on the whole boundary $\partial\mathscr{D}$. The domain has the shape of a head with two eyes, one nose and one mouth. The geometry being known, as well as the initial and boundary conditions, the heat capacity and the time-dependent heat source, the inverse heat conduction problem consists in estimating, through infra-red like temperature measurements on $\mathscr{D}\times\mathscr{I}$, the set of conductivities λ_i, $i=1,\cdots,4$ (1, 2, 3, 4 corresponding to the left eye, the right eye, the noze and the mouth, respectively). Noisy (1 % white noise) synthetic data was performed with conductivities equal to 20, 30, 40 and 50, respectively. Guessed conductivities were all equal to 10.

If optimization methods based on sensitivities are chosen, one will have to compute, sucessively:

$$\rho C\frac{\partial T'}{\partial t}-\nabla\cdot\left(\lambda T'\right)=\nabla\cdot\left(\lambda'\nabla T\right) \tag{89}$$

with null initial and boundary conditions. In the sensitivity model, the direction λ' equals 0 or 1 depending on the location for the four considered sensitivity problems. Corresponding sensitivities are prensented in Figure 11.

With so few parameters to identify (4 in total in this example), it wouldn't be a good idea to use the adjoint-state method to compute the cost gradient. We however give in Figure 11 the evolution of the adjoint state variable which is solved backwardly from final time to initial time while integrating along time the errors integrated within the cost function (this first application was actually chosen for this purposes : small number of unknowns but possible visualization).

From the knowledge of these temperature sensitivities, one can compute the sensitivity matrix S such that $s_{i\times k,j}$ gathers for instance the sensitivity of temperature on the (finite element) node i at time k with respect to λ_j. In the same matter, the error vector $e_{i\times k}$ gathers the error (difference between prediction and measurement) at the (FE) node i and at time t_k. Consequently, the cost gradient is computed straitforwardly through $\nabla j = S^\top e$, and the Gauss–Newton algorithm can be used without any regularization because this parametric problem is not ill-posed. Very few Gauss–Newton iterations are needed to converge as can be seen in Figure 12.

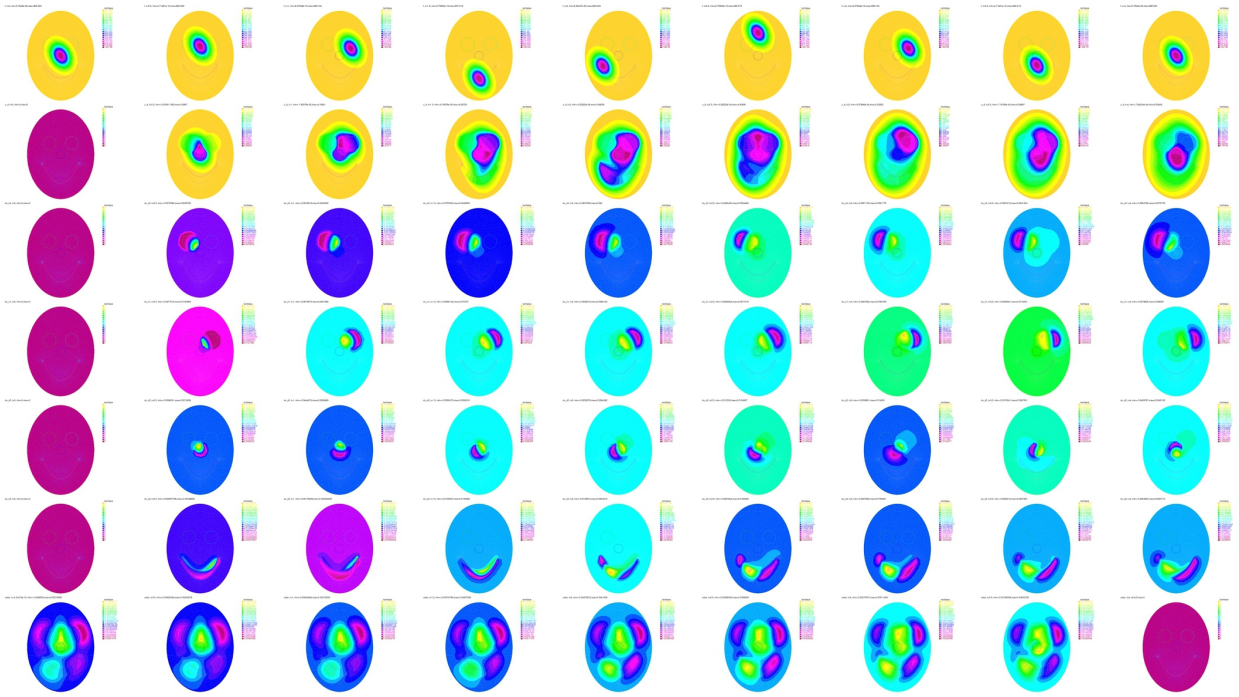


Figure 11: Variables needed to solve the parametric inverse transient heat conduction problem. Columns correspond to increasing time from 0 to 40 by steps of 5. The first raw presents the source term that follows in this particular case a lemniscate. The second raw presents the temperature evolution. The four following raws present the evolutions of sensitivities with respect to λ_1 (the left eye), λ_2 (the right eye), λ_3 (the noze), and λ_4 (the mouth), respectively. The last raw presents the evolution of the adjoint state : it is null at final time and then increases due to cost function integration while going back to initial time.

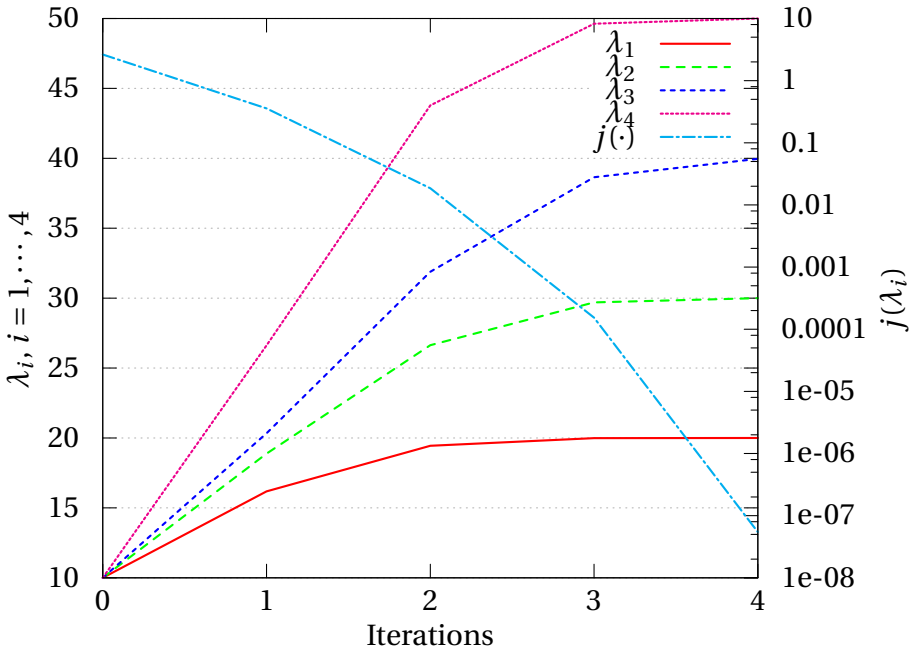


Figure 12: Evolution of the parameters and of the cost function with respect to the G-N iterations.

9.2 Space-dependent convective coefficeint in a transient process

In this section, we consider an application of a nonlinear transient heat transfer inverse problem arising in thermal treatment for instance. $\mathscr{D}$ being an open bounded set of $\mathbb{R}^2$ and $\mathscr{I} =]0, t_f]$, the modeling equation in $\mathscr{D} \times \mathscr{I}$ is

$$C\frac{\partial T}{\partial t} - \nabla \cdot (\lambda \nabla T) = f \quad \text{for } (x,t) \in \mathscr{D} \times \mathscr{I} \tag{90}$$

where temperature dependent physical properties are considered. We also consider the following initial and boundary conditions with $\partial\mathscr{D}_1 \oplus \partial\mathscr{D}_3 \oplus \partial\mathscr{D}_3$ forming a partition of $\partial\mathscr{D}$:

$$\begin{array}{rll} T & = T_0 & \text{at } t = 0 \\ -\lambda \nabla T \cdot \boldsymbol{n} & = h(T - T_\infty) & \text{for } \boldsymbol{x} \in \partial\mathscr{D}_1 \\ \nabla T \cdot \boldsymbol{n} & = 0 & \text{for } \boldsymbol{x} \in \partial\mathscr{D}_2 \\ -\lambda \nabla T \cdot \boldsymbol{n} & = \varepsilon\sigma\left(T^4 - T_\infty^4\right) & \text{for } \boldsymbol{x} \in \partial\mathscr{D}_3 \end{array} \tag{91}$$

The estimation of the heat transfer function $h(\boldsymbol{x}, t)$, $\boldsymbol{x} \in \partial\mathscr{D}_1$, $t \in]0, t_f]$ is performed through the minimization of the cost function:

$$j(h) := \mathscr{J}(T) = \int_0^{t_f} \sum_{j=1}^{N} (T(\boldsymbol{x}_j, t) - T_d(\boldsymbol{x}_j, t))^2 dt \tag{92}$$

where $T(\boldsymbol{x}_j, t)$ and $T_d(\boldsymbol{x}_j, t)$ represent respectively the predicted and measured temperatures at N various locations $\boldsymbol{x} := (r_j, z_j)$ in the material. For such application, the minimization can be carried out by using conjugate gradients or better quasi-Newton methods. In any case, the optimization is based on the gradient computation.

The cost function gradient is obtained for all values $\boldsymbol{x} \in \partial\mathscr{D}_1,\ t \in]0, t_f]$ by the following relationship:

$$\nabla j(h) = T^* \times (T - T_\infty) \tag{93}$$

where T^* is the solution of the adjoint problem:

$$-C\frac{\partial T^*}{\partial t} - \nabla \cdot \left(\lambda \nabla T^*\right) = \sum_j (T - T_d) \times \delta\left(\boldsymbol{x} - \boldsymbol{x}_j\right) \tag{94}$$

with the condition $T^* = 0$ at final time t_f and the conditions on the boundaries:

$$\begin{array}{rl} -\lambda \nabla T^* \cdot \boldsymbol{n} = hT^* & \text{for } \boldsymbol{x} \in \partial\mathscr{D}_1 \\ \nabla T^* \cdot \boldsymbol{n} = 0 & \text{for } \boldsymbol{x} \in \partial\mathscr{D}_2 \\ -\lambda \nabla T^* \cdot \boldsymbol{n} = 4\varepsilon\sigma T^3 T^* & \text{for } \boldsymbol{x} \in \partial\mathscr{D}_3 \end{array} \tag{95}$$

Remark. *The following notations are used:* $\mathscr{U} := L_2(\mathscr{I}; L_2(\mathscr{D}))$, $\mathscr{U}_i := L_2(\mathscr{I}; L_2(\partial\mathscr{D}_i))$, $\forall i = 1,2,3$ *and* $\mathscr{U}_* := L_2(\mathscr{I}; L_2(\cup_i \partial\mathscr{D}_i))$.

Proof. The derivative of the state T at the point h and towards δh, $T'(h;\delta h)$ is defined by:

$$\left\{\begin{array}{ll} C\frac{\partial T'}{\partial t} - \nabla \cdot \lambda \nabla T' = 0 & \boldsymbol{x} \in \mathscr{D},\ t \in \mathscr{I} \\ T' = 0 & \boldsymbol{x} \in \mathscr{D},\ t = 0 \\ \lambda \nabla T' \cdot \boldsymbol{n} + hT' + \delta h(T - T_\infty) = 0 & \boldsymbol{x} \in \partial\mathscr{D}_1,\ t \in \mathscr{I} \\ \nabla T' \cdot \boldsymbol{n} = 0 & \boldsymbol{x} \in \partial\mathscr{D}_2,\ t \in \mathscr{I} \\ \lambda \nabla T' \cdot \boldsymbol{n} + 4\varepsilon\sigma T^3 T' = 0 & \boldsymbol{x} \in \partial\mathscr{D}_3,\ t \in \mathscr{I} \end{array}\right. \tag{96}$$

The Lagrange function is formely defined as:

$$\begin{array}{rl} \mathscr{L}(T, \{T^*, \gamma, \xi, \varpi\}, h) = & \mathscr{J}(T) + \left(C\frac{\partial T}{\partial t} - \nabla \cdot (\lambda \nabla T) - f, T^*\right)_{\mathscr{U}} \\ & + \left(\lambda \nabla T \cdot \boldsymbol{n} + h\left(T - T_\infty\right), \gamma\right)_{\mathscr{U}_1} \\ & + \left(\nabla T \cdot \boldsymbol{n}, \xi\right)_{\mathscr{U}_2} \\ & + \left(\lambda \nabla T \cdot \boldsymbol{n} + \varepsilon\sigma\left(T^4 - T_\infty^4\right), \varpi\right)_{\mathscr{U}_3} \end{array} \tag{97}$$

The differentiated Lagrange function with respect to h is the direction δh is:

$$\begin{array}{rl} \left(\mathscr{L}'_h(\cdot), \delta h\right) = & \left(T - T_d, T'\right)_{\mathscr{X}} \\ & + \left(C\frac{\partial(T')}{\partial t} - \nabla \cdot \lambda \nabla T', T^*\right)_{\mathscr{U}} \\ & + \left(\lambda \nabla T') \cdot \boldsymbol{n} + hT' + \delta h(T - T_\infty), \gamma\right)_{\mathscr{U}_1} \\ & + \left(\nabla T' \cdot \boldsymbol{n}, \xi\right)_{\mathscr{U}_2} \\ & + \left(\lambda \nabla T' \cdot \boldsymbol{n} + 4\varepsilon\sigma T^3 T', \varpi\right)_{\mathscr{U}_3} \end{array} \tag{98}$$

We then use the following integrations by parts to express some particular terms:

$$\begin{array}{rl} \left(C\frac{\partial T'}{\partial t}, T^*\right)_{\mathscr{U}} = & \left(T', -C\frac{\partial T^*}{\partial t}\right)_{\mathscr{U}} + \left(CT', T^*\right)_{\mathscr{D}}(t = t_f) - \left(CT', T^*\right)_{\mathscr{D}}(t = 0) \\ \left(\lambda \Delta T', T^*\right)_{\mathscr{U}} = & \left(\lambda \Delta T^*, T'\right)_{\mathscr{U}} + \left(\lambda \nabla T^* \cdot \boldsymbol{n}, T'\right)_{\mathscr{U}_*} - \left(\lambda T^*, \nabla T' \cdot \boldsymbol{n}\right)_{\mathscr{U}_*} \end{array} \tag{99}$$

We bring together similar terms to get:

$$\begin{array}{rl} \left(\mathscr{L}'_h(T, \{T^*, \gamma, \xi, \varpi\}, h), \delta h\right) = & \left(T - T_d, T'\right)_{\mathscr{X}} + \left(\delta h\left(T - T_\infty\right), \gamma\right)_{\mathscr{U}_1} \\ & + \left(-C\frac{\partial T^*}{\partial t} - \lambda \Delta T^*, T'\right)_{\mathscr{U}} + \left(CT^*, T'\right)_{\mathscr{D}}(t = t_f) \\ & + \left(\lambda \nabla T^* \cdot \boldsymbol{n}, T'\right)_{\mathscr{U}_*} - \left(\lambda T^*, \nabla T' \cdot \boldsymbol{n}\right)_{\mathscr{U}_*} \\ & + \left(\lambda \nabla T') \cdot \boldsymbol{n} + hT', \gamma\right)_{\mathscr{U}_1} + \left(\nabla T' \cdot \boldsymbol{n}, \xi\right)_{\mathscr{U}_2} \\ & + \left(\lambda \nabla T' \cdot \boldsymbol{n} + 4\varepsilon\sigma T^3 T', \varpi\right)_{\mathscr{U}_3} \end{array} \tag{100}$$

Choosing $\gamma = T^*$ on $\partial\mathscr{D}_1$, $\xi = \lambda T^*$ on $\partial\mathscr{D}_2$ and $\varpi = T^*$ on $\partial\mathscr{D}_3$, the adjoint problem can eventually be written as:

$$\left\{\begin{array}{ll} -C\frac{\partial T^*}{\partial t} - \lambda\Delta T^* = -\sum_j (T - T_d)\times\delta(\boldsymbol{x}-\boldsymbol{x}_j) & \boldsymbol{x}\in\mathscr{D},\ t\in\mathscr{I} \\ T^* = 0 & \boldsymbol{x}\in\mathscr{D},\ t = t_f \\ -\lambda\nabla T^*\cdot\boldsymbol{n} = hT^* & \boldsymbol{x}\in\partial\mathscr{D}_1,\ t\in\mathscr{I} \\ \quad \nabla T^*\cdot\boldsymbol{n} = 0 & \boldsymbol{x}\in\partial\mathscr{D}_2,\ t\in\mathscr{I} \\ -\lambda\nabla T^*\cdot\boldsymbol{n} = 4\epsilon\sigma T^3 T^* & \boldsymbol{x}\in\partial\mathscr{D}_3,\ t\in\mathscr{I} \end{array}\right. \tag{101}$$

and the cost gradient is written as:

$$\nabla j = -\left(T - T_\infty\right) T^*. \tag{102}$$

□

From the integration of the adjoint-state, the cost function gradient is computed. From the knowledge of the cost function gradient, the direction of descent is computed, for instance with the conjugate gradient method, or with any other faster method if a fine parameterization for h is required. Next, the temperature state being varying almost linearly with the heat flux, the line-search equation can be for instance given by the solution of (14).

9.3 Adjoint RTE

Let the bi-dimensional radiative transfer equation (RTE) being written as, $\forall\,(\boldsymbol{x},\boldsymbol{s})\in\mathscr{D}\times 2\pi$:

$$(\boldsymbol{s}\cdot\nabla + \kappa + \sigma)\, L(\boldsymbol{x},\boldsymbol{s}) = \sigma \oint_{2\pi} L(\boldsymbol{x},\boldsymbol{s})\Phi(\boldsymbol{s},\boldsymbol{s}')\,\mathrm{d}\omega(\boldsymbol{s}') \tag{103}$$

where $\boldsymbol{s}$ is the considered direction of propagation, $\Phi(\boldsymbol{s},\boldsymbol{s}')$ is the phase function representing the probability that a photon arriving from the direction $\boldsymbol{s}'$ is scattered to the direction $\boldsymbol{s}$, and κ and σ are the absorption and diffusion space-dependent functions, respectively. On a part of the boundary, there is a prescribed Dirichlet boundary condition:

$$L(\boldsymbol{x},\boldsymbol{s}) = L^0 \quad \text{for } \boldsymbol{x}\in\partial\mathscr{D}_s \text{ and } \boldsymbol{s} = \boldsymbol{s}_0 \tag{104}$$

Also, let a cost function measuring the misfit between predictions and measurements somewhere on the boundary, i.e. $\partial\Omega_d \subset \partial\Omega$, the misfit being expressed (it is actually a norm) in terms of the radiance,

$$e = L(\boldsymbol{x},\boldsymbol{s}) - L_d(\boldsymbol{x},\boldsymbol{s}) \quad \text{for } \boldsymbol{x}\in\partial\mathscr{D}_d \text{ and } \boldsymbol{s} = \boldsymbol{s}_d \tag{105}$$

In order to make the derivation of the adjoint-state, the tools described in previous sections are used. Additionally, the state variable L being defined in $(\boldsymbol{x},\boldsymbol{s})\in\mathscr{D}\times 2\pi$, the inner product $\mathscr{U}$ defined in (64) and in following equations is:

$$(u,v)_{\mathscr{U}} = \int_{2\pi}\int_{\mathscr{D}} uv\,\mathrm{d}\boldsymbol{x}\mathrm{d}\boldsymbol{s} \tag{106}$$

After integrations by parts and manipulations in inner products, one finds the adjoint RTE to be:

$$(-\boldsymbol{s}\cdot\nabla + \kappa + \sigma)\, L^*(\boldsymbol{x},\boldsymbol{s}) = \sigma \oint_{2\pi} L*(\boldsymbol{x},\boldsymbol{s})\Phi(\boldsymbol{s},\boldsymbol{s}')\,\mathrm{d}\omega(\boldsymbol{s}') \tag{107}$$

coupled with the Dirichlet boundary condition:

$$L^*(\boldsymbol{x},\boldsymbol{s}) = (\boldsymbol{s}\cdot\boldsymbol{n})^{-1}(L - L_d)(\boldsymbol{x},\boldsymbol{s}) \quad \text{for } \boldsymbol{x} \in \partial\mathscr{D}_d \text{ and } \boldsymbol{s} = \boldsymbol{s}_d \tag{108}$$

The Figure 13 presents the foward photon density for a 4 tests configuration and related adjoint versions.

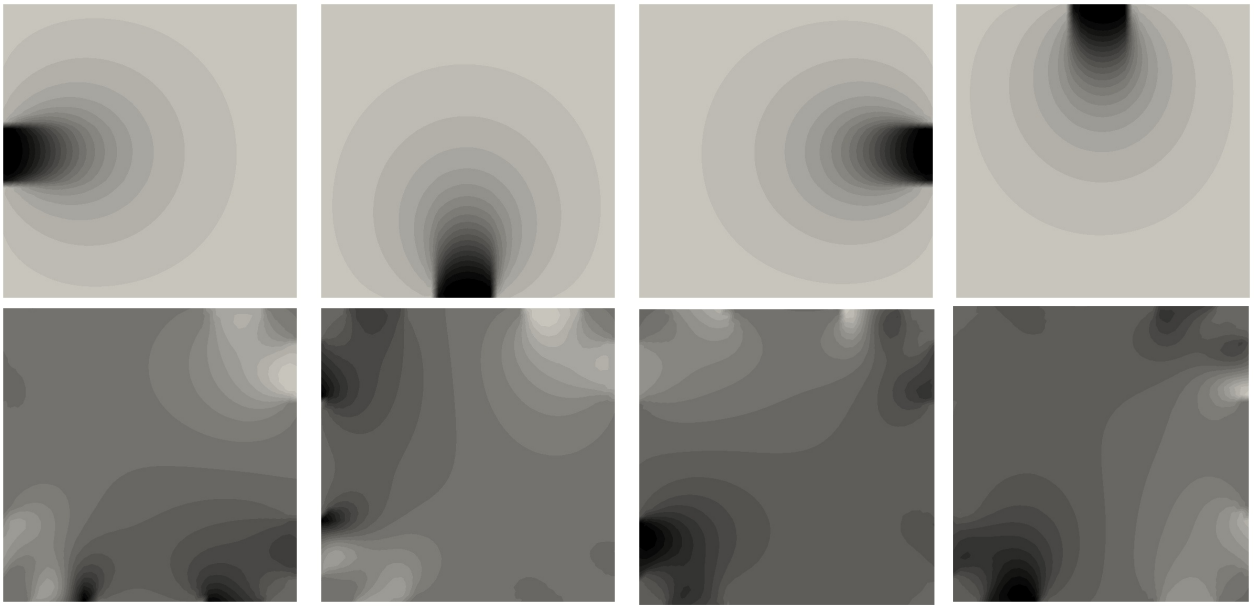


Figure 13: Top: foward photon density $\Phi(\boldsymbol{x}) = \int_{2\pi} L(\boldsymbol{x},\boldsymbol{s})$ with the Dirichlet BC acting as a source. Bottom: backward adjoint photon density $\Phi^*(\boldsymbol{x}) = \int_{2\pi} L^*(\boldsymbol{x},\boldsymbol{s})$ with the misfit acting as the source in the reverse directions. From left to right: 1st source, 2nd source, 3rd source and 4th source.

References

[1] G. Allaire. *Analyse numérique et optimisation.* Les Éditions de l'École Polytechnique, Paris, Août 2005.

[2] M. Minoux. *Programmation mathématique : théorie et algorithms, tomes 1 et 2.* Dunod, Paris, 1983.

[3] M. Minoux. *Mathematical Programming, Theory and Applications.* Wiley, Chichester, UK, 1986.

[4] J. Nocedal and S.J. Wright. *Numerical Optimization.* Springer, New York, 1999.

[5] P.E. Gill, W. Murray, and M.H. Wright. *Practical Optimization.* Academic Press, London, 1992.

[6] Y. Favennec. Lecture C3b: Optimization, sensitivity and adjoint states. Eurotherm Winter School METTI 2005 : Thermal measurements and inverse techniques: a tool for the characterization of multiphysical phenomena, Aussois, 2005.

[7] Y. Favennec, P. Le Masson, and Y. Jarny. Lecture 7: Optimization methods for non linear estima- tion or function estimation. Eurotherm School METTI 2011 : Thermal measurements and inverse techniques, Roscoff, 2011.

[8] O. Alifanov. *Inverse Heat Conduction, Ill-posed problems.* Wiley Interscience, New-York, 1985.

[9] G.C. OnWubolu and B.V. Babu. *New optimization techniques in engineering.* Springer, 2003.

[10] M. Clerc. *L'optimisation par essaims particulaires.* Hermes-Lavoisier, 2005.

[11] M. Clerc. http://clerc.maurice.free.fr/pso/.

[12] J.C. Culioli. *Introduction à l'optimisation.* Ellipses, Paris, 1994.

[13] J. Céa. *Optimisation, théorie et algorithmes.* Dunod, Paris, 1971.

[14] B. Larrouturou and P.L. Lions. *Méthodes mathématiques pour les sciences de l'ingénieur: optimisation et analyse numérique.* Cours de l'École Polytechnique, Paris, 1994.

[15] O Balima, Y Favennec, and D Rousse. Optical tomography reconstruction algorithm with the finite element method: An optimal approach with regularization tools. *Journal of Computational Physics*, 251:461–479, 2013.

[16] Y Favennec, F Dubot, B Rousseau, and D Rousse. Mixing regularization tools for enhancing regularity in optical tomography applications. In O Fudym, J L Battaglia, and G S Dulikravich, editors, *IPDO 2013 : 4th Inverse problems, design and optimization symposium*, Albi, June 26-28 2013.

[17] Fabien Dubot, Yann Favennec, Benoit Rousseau, and Daniel R. Rousse. A wavelet multiscale method for the inverse problem of diffuse optical tomography. *Journal of Computational and Applied Mathematics*, 2015.