Lecture 7: Optimization methods for non linear estimation or function estimation

Y. Favennec[1], **P. Le Masson** [2] **and Y. Jarny**[1]

[1] LTN – UMR CNRS 6607 – Polytetch' Nantes – 44306 Nantes – France

[2] LIMATB – Université de Bretagne Sud – 56325 Lorient – France

E-mail: `yann.favennec@univ-nantes.fr`

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. 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

1. Introduction

This lecture is devoted to the use of optimization methods for the solution of non-linear 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 "Zero-order optimization" [1].

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 [2, 3, 4, 5, 6]. 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 the lecture "C3b" given in a previous Eurotherm Metti school [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 "direct" 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 is nowadays 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. Gauss–Newton or Levenberg–Marquardt 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 a physical coefficient is state-dependent, and if the state depends on, for instance, $\boldsymbol{x} \in \Omega$, then the coefficient also depends on $\boldsymbol{x}$, so that one may identify $\lambda = \lambda(\boldsymbol{x})$. If some (for instance finite element) discretization is used to approach the continuous-state solution, then the function can also be discretized –actually parametrized– as above with the form $\lambda = \sum \lambda_i \xi^i(\boldsymbol{x})$ and then, as above, one searches the λ_i; one deals again 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 parametrized, so that we turn out to be back to parameter estimation.

Maybe the main 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, see Lecture 9 [8], 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 parametrization must not be "as finer as possible" for regularity considerations, and as recalled in Lecture 3 [9].

2.2. The function to be minimized

In inversion process, one usually minimizes a discrepancy between some experimental data (say u_d noted y in some other Metti Lectures) and some model data (say u noted y_{mo} in some other Metti Lectures). The discrepancy function (also called cost 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. $\|u - u_d\|_{L_2(\mathcal{S})} = \int_{\mathcal{S}}(u - u_d)\,\mathrm{d}s$ 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: $\|u - u_d\|_2^2 := \sum_i (u^i - u_d^i)^2 = \int_{\mathcal{S}} \delta_i^j (u - u_d)\,\mathrm{d}s$. 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 can also be considered. So, in order to write down a general form for the cost function to be minimized, we use :

$$\mathcal{J}(u) = f\left(\|u - u_d\|^2\right) \tag{1}$$

without specifying any choice for the norm. The norm $\|\cdot\|$ is squared so that the function $\mathcal{J}$ does not a priori present any discontinuity. If some regularization terms are added to the cost function, then one uses:

$$\mathcal{J}(u, \psi) = f\left(\|u - u_d\|^2\right) + g\left(\|\psi\|^2\right) \tag{2}$$

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 is searched, i.e. the parameters ψ. Hence we write the equality (by definition):

$$j(\psi) := \mathcal{J}(u) \tag{3}$$

where the function j is often called the reduced cost function, as opposed to $\mathcal{J}$ which is the 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 (3) is defined on $\mathcal{K}$ with values in $\mathbb{R}$. $\mathcal{K}$ is a set of admissible elements of the problem. In some cases, $\mathcal{K}$ defines some constraints on the parameters or functions. The minimization problem is written as:

$$\inf_{\phi \in \mathcal{K} \subset \mathcal{V}} j(\phi). \tag{4}$$

According to [2], 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 $\phi \in \mathcal{K}$ such that

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

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

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

Let us now recall basic definitions needed for mathematical optimization [2]:

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

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

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

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

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

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

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

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

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

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 $\mathcal{K}$, the local minimum of j on $\mathcal{K}$ is the global minimum on $\mathcal{K}$.

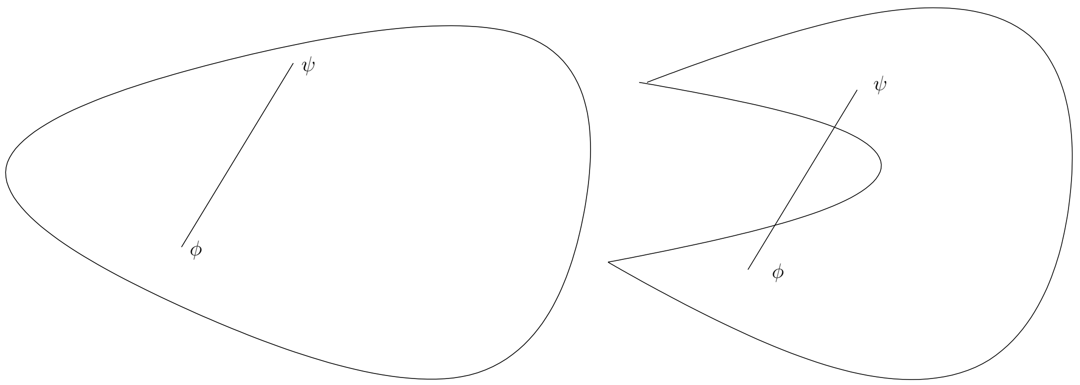

Figure 1. Convex and non-convex domaine $\mathcal{K}$

2.4. Optimality conditions

For convex functions, there is no difference between local minima and global minimum. In the sequel 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 the gradient-free optimization algorithms may find the global minimum while the 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:

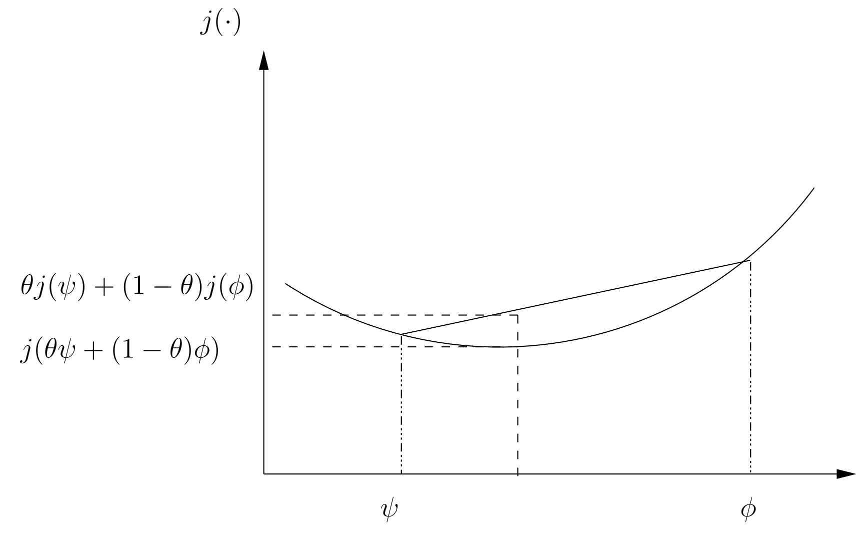
$j(\cdot)$
$\theta j(\psi) + (1-\theta) j(\phi)$
$j(\theta\psi + (1-\theta)\phi)$
ψ
ϕ

Figure 2. Convex function

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 minimum. 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.

$$\left\|\nabla j(\psi^p)\right\|_\infty \leq \varepsilon; \tag{5}$$
$$\left\|\nabla j(\psi^p)\right\|_2 \leq \varepsilon; \tag{6}$$
$$\left|j(\psi^p) - j(\psi^{p-1})\right| \leq \varepsilon; \tag{7}$$
$$\psi^p - \psi^{p-1} \leq \varepsilon; \tag{8}$$
$$j(\psi^p) \leq \varepsilon. \tag{9}$$

For each of the above criteria, it may be judicious to demand that the test be satisfied over several successive iterations. The four first above-presented criteria are convergence criteria applied on the cost function gradient, on the cost function value itself, or on the parameters: the first two criteria are the $\|\cdot\|_\infty$ and $\|\cdot\|_2$ norms of the cost function gradient at iteration p; the third criterion is related to the stabilization of the cost function from the actual iteration with respect to the previous one, and the fourth criterion is related to the stabilization of the parameters. 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 [10]. It will be seen in the section "Examples" at the end of this lecture that the consequence of lowering the cost function below a given criterion based on measurement errors only affects the result in highliting its inherent noise.

3. Zero-order n–dimensional optimization algorithms

Zero-order methods, also called Derivative-free optimization (DFO) are based on a global vision of the cost function value j. 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 [11].

In the sequel, we restrict ourself in very briefly presenting a deterministic algoritm, the so-called simplex method, and one probabilistic method, 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 $\mathscr{S}_0 = \left\{\psi^I, I = 1, \ldots, n+1\right\}$. 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 (see Figure 3). Let $\bar{\psi}$ the isobarycenter of $\left\{\psi^I, I = 1, \ldots, n, \right\}$ (without ψ^h) and let the ordering so that

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

and let $\psi^\ell = \arg_{I=1,\ldots,n} \min j(\psi^I)$ and $\psi^h = \arg_{I=1,\ldots,n} \max j(\psi^I)$.

At each iteration, the simplex improvement is performed in three steps:

(i) [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);

(ii) [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;

(iii) [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 $\forall I \neq L$ ψ^I by $(\psi^I + \psi^\ell)/2$.

The basic operations for $n = 2$ are given in figure 3.

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 $\mathcal{K}$ interacting with other particles on their best locations. From this

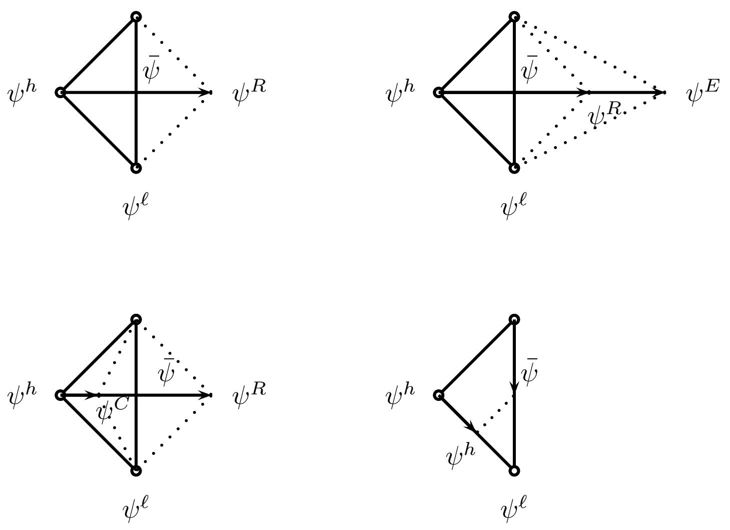

Figure 3. Basic operations on a simplex for $n = 2$. Top left: reflection; top right: expansion; bottom left : contraction; bottom right : inside contraction.

information, the particle shall change its position ψ^i and its velocity $\delta\psi^i$. The general formulation for this behavior is given by:

$$\begin{array}{l} \delta\psi^i = \chi\delta\psi^i + \lambda_1 \mathrm{rand}_1 \left(\phi^g - \psi^i\right) + \lambda_2 \mathrm{rand}_2 \left(\phi^i - \psi^i\right) \\ \psi^i = \psi^i + \delta\psi^i \end{array} \tag{10}$$

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 4). χ, λ_1 and λ_2 are some coefficients weighting the three directions of the particule [11]:

- 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 [12]:

- swarm size, usually between 20 and 30;
- initialization of both the position of the particles and their velocity $\sim \mathcal{U}[0, 1]$,
- 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 [13].

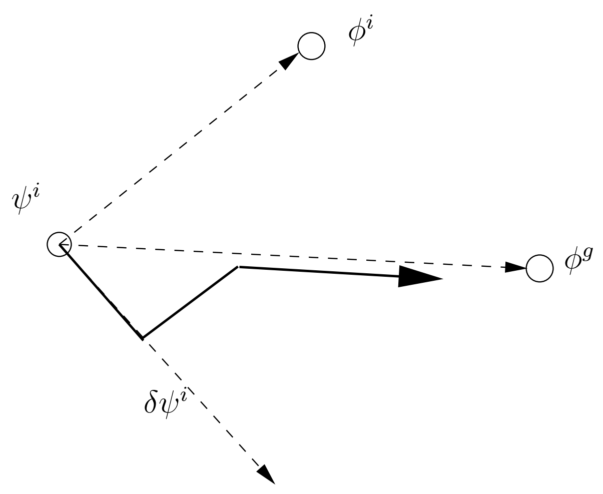

Figure 4. PSO algorithm: a particle displacement.

4. One-dimensional unconstrained opimization

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:

$$\text{Find } \bar{\alpha} = \arg\min_{\alpha>0} g(\alpha) = j\left(\psi^p + \alpha d^p\right), \tag{11}$$

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.

However, one has to keep in mind that the main objective is not to solve (11) but to find the minimum of $j(\psi)$ through the minimum of $\mathcal{J}(u)$. Thus one has to design an efficient tool for the one-dimensional algorithm that finds the minimum of $g(\alpha)$ or rather approach it (inexact line search algorithm).

Note that we always assume that $g'(0) = (\nabla j(\psi^p), d^p) < 0$ meaning that d^p is indeed a descent direction.

4.1. 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 nonlinear 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 $g'\left(\alpha^{q+1}\right) = g'\left(\alpha^q\right) + g''\left(\alpha^q\right).\left(\alpha^{q+1} - \alpha^q\right) = 0$ which gives:

$$\alpha^{q+1} = \alpha^q - \frac{g'\left(\alpha^q\right)}{g''\left(\alpha^q\right)}. \tag{12}$$

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 [3]. 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.

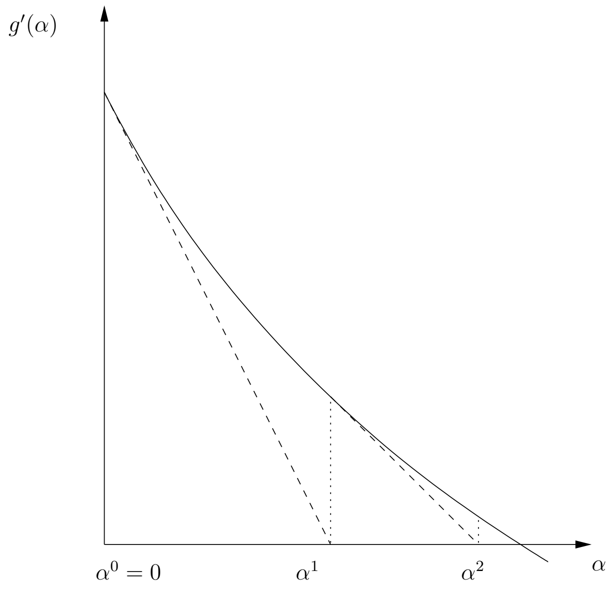

Figure 5. The Newton–Raphson line search method.

4.2. The secant method

The second-order derivative $g''(\alpha)$ is approximated by (13) so that the Newton–Raphson's equation given by (12) becomes (14):

$$g''(\alpha^q) \approxeq \frac{g'(\alpha^q) - g'\left(\alpha^{q-1}\right)}{\alpha^q - \alpha^{q-1}}; \tag{13}$$

$$\alpha^{q+1} \approxeq \alpha^q - g'(\alpha^q)\frac{\alpha^q - \alpha^{q-1}}{g'(\alpha^q) - g'(\alpha^{q-1})}. \tag{14}$$

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 $[\alpha^q, g'(\alpha^q)]$.

4.3. The quadratic interpolation

By comparison of those of sections 4.1 and 4.2, 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:

$$\alpha_4 = \frac{1}{2}\,\frac{r_{23}g(\alpha_1) + r_{31}g(\alpha_2) + r_{12}g(\alpha_3)}{s_{23}g(\alpha_1) + s_{31}g(\alpha_2) + s_{12}g(\alpha_3)}, \tag{15}$$

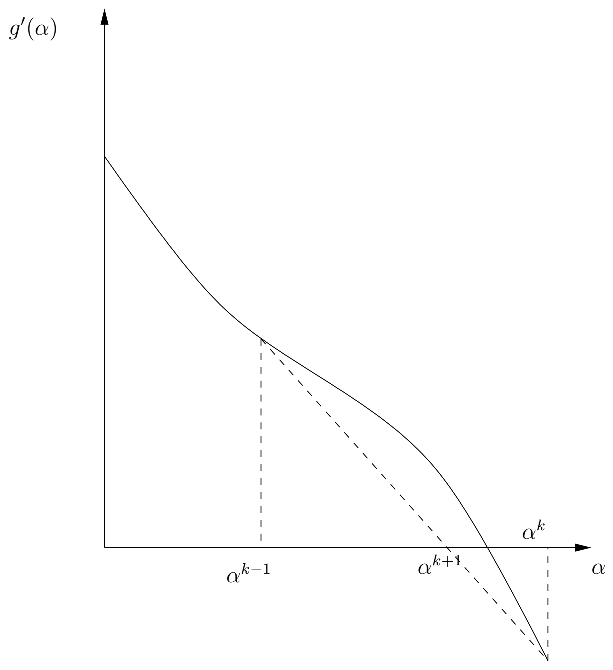

Figure 6. The secant line search method.

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 [3].

4.4. 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 [3].

4.5. Other methods

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 both the Fibonacci method and the golden section search method are very widely used. The cubic interpolation method is also very widely used in practice. The reader may refer to [6, 4, 3] for more details.

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

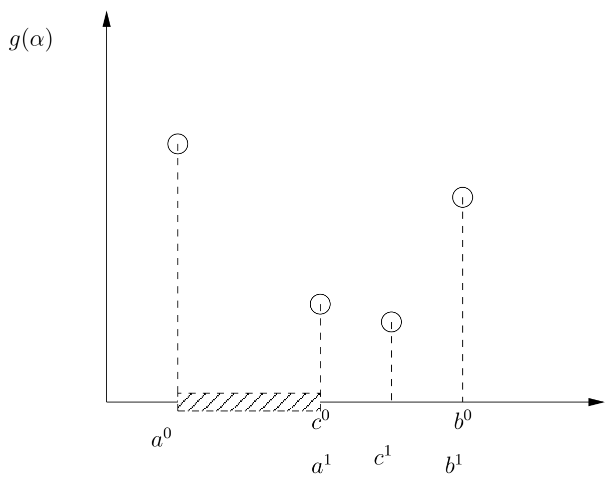

Figure 7. The dichotomy line search method.

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 ($d^p = -\nabla j(\psi^p)$), or computed from the gradient or chosen in another way provided that it is a descent direction, i.e. $(\nabla j(\psi^p), d^p) < 0$.

5.1. 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 is thus to compute the gradient, and find the new point according to the predefined strictly positive step size α^p as:

$$\psi^{p+1} = \psi^p - \alpha^p \frac{\nabla j(\psi^p)}{\|\nabla j(\psi^p)\|}. \tag{16}$$

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.

5.2. 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 may be very high (see figure 8 and then figure 13 page 20).

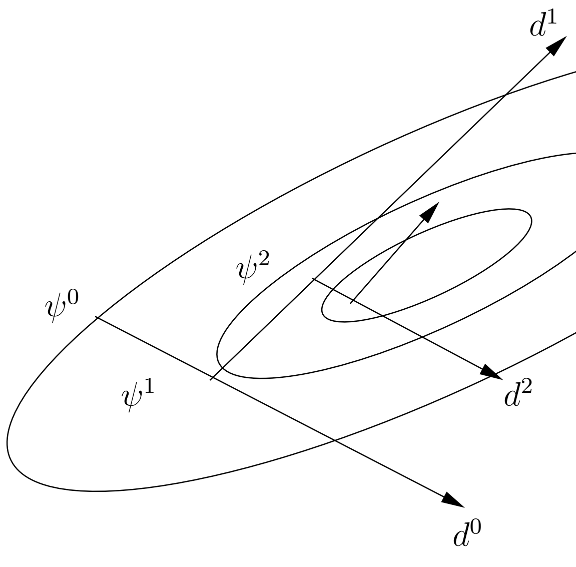

Figure 8. When the steepest descent method is used, the two consecutive directions are orthogonal.

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

In this section we shall firstly assume that the cost function is quadratic. We shall then develop the general algorithm usually applied on quadratic functions. The case of arbitrary functions is dealt with in the next subsection. Let the quadratic functional be of the form:

$$j(\psi) = \frac{1}{2}\left(\mathscr{A}\psi, \psi\right), \tag{17}$$

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:

$$\left(d^0, \nabla j\left(\psi^1\right)\right) = 0. \tag{18}$$

Using the relationship $\nabla j(\psi) = \mathscr{A}\psi$ given by the differentiation of (17) and the reactualization formulation $\psi^1 = \psi^0 + \alpha^0 d^0$, (18) becomes:

$$\begin{aligned} \left(d^0, \nabla j\left(\psi^1\right)\right) &= \left(d^0, \mathscr{A}\psi^1\right) \\ &= \left(d^0, \mathscr{A}\left(\psi^0 + \alpha^0 d^0\right)\right) \\ &= \left(d^0, \mathscr{A}\psi^0\right) + \alpha^0\left(d^0, \mathscr{A}d^0\right). \end{aligned} \tag{19}$$

Equaling (19) to zero gives the step size α^0:

$$\alpha^0 = -\frac{\left(d^0, \mathscr{A}\psi^0\right)}{\left(d^0, \mathscr{A}d^0\right)}. \tag{20}$$

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:

$$\begin{aligned}\left(d^p, \mathscr{A} d^{p-1}\right) &= \left(-\nabla j\left(\psi^p\right) + \beta^p d^{p-1}, \mathscr{A} d^{p-1}\right) \\ &= -\left(\nabla j\left(\psi^p\right), \mathscr{A} d^{p-1}\right) + \beta^p \left(d^{p-1}, \mathscr{A} d^{p-1}\right).\end{aligned} \tag{21}$$

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

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

Taking into account of above relationships, the *conjugate gradient algorithm* [3] is given in Algorithm 1.

Algorithm 1: The conjugate gradient algorithm applied on quadratic functions

```
(i)   Let $p = 0$, $\psi^0$ be the starting point,
      compute the gradient and the descent direction, $d^0 = -\nabla j(\psi^0)$,
      compute the step size $\alpha^0 = -\frac{\left(d^0, \mathscr{A}\psi^0\right)}{\left(d^0, \mathscr{A} d^0\right)}$;
(ii)  At step $p$, we are at the point $\psi^p$.
      We define $\psi^{p+1} = \psi^p + \alpha^p d^p$ with:
        • the step size $\alpha^p = -\frac{(d^p, \nabla j(\psi^p))}{(d^p, \mathscr{A} d^p)}$
        • the direction $d^p = -\nabla j(\psi^p) + \beta^p d^{p-1}$
        • where the coefficient needed for conjugate directions: $\beta^p = \frac{\left(\nabla j(\psi^p), \mathscr{A} d^{p-1}\right)}{\left(d^{p-1}, \mathscr{A} d^{p-1}\right)}$;
(iii) Stopping rule (see subsection 2.5). If satisfies: End, otherwise set $p \leftarrow p + 1$ and return to
      step (ii).
```

It may be proved [4] that the conjugate gradient method applied on quadratic functions converges in at most n iterations where $n = \dim \psi$.

5.4. 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 (17) and taking into account of the reactualization relationship gives directly:

$$\begin{aligned}\nabla j(\psi^p) - \nabla j(\psi^{p-1}) &= \mathscr{A}\left(\psi^p - \psi^{p-1}\right) \\ &= \mathscr{A}\left(\psi^{p-1} + \alpha^{p-1} d^{p-1} - \psi^{p-1}\right) \\ &= \alpha^{p-1} \mathscr{A} d^{p-1},\end{aligned} \tag{23}$$

which also gives the following relationship:

$$\frac{1}{\alpha^{p-1}}\left(\nabla j(\psi^p), \nabla j(\psi^p) - \nabla j(\psi^{p-1})\right) = \left(\nabla j(\psi^p), \mathscr{A} d^{p-1}\right). \tag{24}$$

On the other hand, substituting (24) into (22) gives

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

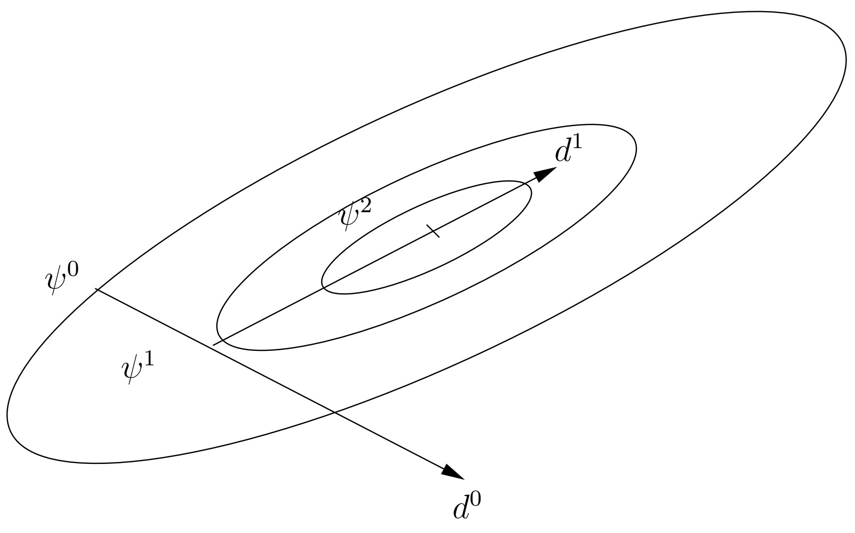

Figure 9. When the conjugate gradient method is used, the two consecutive directions are conjugate instead of orthogonal. Applied on a quadratic function, the method converges in at most n iterations (in this figure two iterations are needed since $\dim\psi = 2$).

Next, expanding the descent direction d^{p-1}, (25) becomes:

$$\beta^p = \frac{\big(\nabla j(\psi^p), \nabla j(\psi^p) - \nabla j(\psi^{p-1})\big)}{(-\nabla j(\psi^{p-1}) + \beta^{p-1} d^{p-2}, \nabla j(\psi^p) - \nabla j(\psi^{p-1}))}; \tag{26}$$

$$\beta^p = \frac{\big(\nabla j(\psi^p), \nabla j(\psi^p) - \nabla j(\psi^{p-1})\big)}{(-\nabla j(\psi^{p-1}) - \beta^{p-1}\nabla j(\psi^{p-2}) + \Lambda, \nabla j(\psi^p) - \nabla j(\psi^{p-1}))}, \tag{27}$$

where Λ is the series given from the reactualizations. All the gradients being orthogonal one from the next, (27) becomes:

$$\beta^p = \frac{\big(\nabla j(\psi^p), \nabla j(\psi^p) - \nabla j(\psi^{p-1})\big)}{(\nabla j(\psi^{p-1}), \nabla j(\psi^{p-1}))}, \tag{28}$$

and also:

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

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 applying (29) to access β^p while the Polak and Ribiere's method consists in applying (28) to access β^p. Taking into account of above remarks, the *conjugate gradient algorithm applied on arbitrary functions* is given in Algorithm 2.

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.

5.5. 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 cost function gradient by its quadratic approximation through a Taylor development:

```
Algorithm 2: The conjugate gradient algorithm applied on arbitrary functions
  (i) Let $p = 0$, $\psi^0$ be the starting point, $d^0 = -\nabla j(\psi^0)$;
  (ii) At step $p$, we are at the point $\psi^p$; 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 $\beta^p$ satisfies either (28) or (29) depending on the chosen method;
  (iii) Stopping rule (see subsection 2.5). If satisfies: End, otherwise, set $p \leftarrow p + 1$ and return to step (ii).
```

$$\nabla j(\psi^{p+1}) = \nabla j(\psi^p) + \left[\nabla^2 j(\psi^p)\right] \delta\psi^p + \mathcal{O}\left(\delta\psi^p\right)^2, \tag{30}$$

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 (30) 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, in practice, prefer using one of the numerous quasi-Newton methods detailed afterwards.

5.6. 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 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:

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

so that the approximation given by (31) is valid at previous step $p - 1$. Various corrections of the type

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

may be found in literature [3].

5.7. Rank 1 *correction [3]*
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 $\Delta^p = \alpha^p v^p v^{pt}$ where the vector v^p and the scalar α^p are chosen such that (32) is satisfied. Then, from a symetric matrix $\mathbf{H}^0$, this 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:

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

and

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

thus

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

Using the identity

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

and using (36) and (37) to get

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

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

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

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

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

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

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 3.

Algorithm 3: The Davidon – Fletcher – Powell (DFP) algorithm

(i) Let $p = 0$, ψ^0 be the starting point. Choose any positive definite matrix $\mathbf{H}^0$ (often the identity matrix);

(ii) 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 $\alpha \geq 0$;

(iii) set $\delta^p = \psi^{p+1} - \psi^p$ and compute $\gamma^p = \nabla j(\psi^{p+1}) - \nabla j(\psi^p)$ to actualize:

$$\mathbf{H}^{p+1} = \mathbf{H}^p + \frac{\delta^p (\delta^p)^t}{(\delta^p)^t \gamma^p} - \frac{\mathbf{H}^p \gamma^p (\gamma^p)^t \mathbf{H}^p}{(\gamma^p)^t \mathbf{H} \gamma^p}; \tag{45}$$

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

5.9. The Broyden – Fletcher – Goldfarb – Shanno 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 [3] 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 inverse Hessian $\mathbf{H}^{p+1}$ is thus given from $\mathbf{H}^p$ through the correction formulation:

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

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 [6, 14]. It is known that the BFGS algorithm is superior than the DFP algorithm is the sense that the first one is less sensitive on the line-search inaccuracy than the latter [3].

Algorithm 4: The BFGS algorithm

(i) Let $p = 0$, ψ^0 be the starting point. Choose any positive definite matrix $\mathbf{H}^0$ (often the identity matrix);

(ii) 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 $\alpha \geq 0$;

(iii) set $\delta^p = \psi^{p+1} - \psi^p$ and compute $\gamma^p = \nabla j(\psi^{p+1}) - \nabla j(\psi^p)$ to actualize:

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

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

5.10. Gauss–Newton

When the cost function is explicitly given in term of the state and squared, that is of the form

$$j(\psi) := \mathcal{J}(u) = \int_{\mathcal{S}} (u - u_d)^2 \, \mathrm{d}s \tag{48}$$

where $\mathcal{S}$ may be a time interval or a geometrical interval for instance including pointwise measurements, 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 not very high (see Lecture 3 [9]).

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:

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

then the directional derivative of the cost function writes:

$$j'(\psi;\delta\psi) = \left(\mathcal{J}'(u), u'(\psi;\delta\psi)\right), \tag{50}$$

where $j'(\psi;\delta\psi) = (\nabla j(\psi), \delta\psi)$. 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:

$$j''(\psi;\delta\psi,\delta\phi) = \left(\mathcal{J}'(u), u''(\psi;\delta\psi,\delta\phi)\right) + \left(\left(\mathcal{J}''(u), u'(\psi;\delta\psi)\right), u'(\psi;\delta\phi)\right). \tag{51}$$

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

$$j''(\psi;\delta\psi,\delta\phi) \approx \left(\left(\mathcal{J}''(u), u'(\psi;\delta\psi)\right), u'(\psi;\delta\phi)\right). \tag{52}$$

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 $(u'(\psi;\delta\psi_i), u'(\psi;\delta\psi_j))$ involved in (52) is the product of the so-called sensitivity matrix with its transposed. The Newton relationship is thus approximated to:

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

The matrix system S^tS is obviously symmetric and positive definite with a dominant diagonal yielding thus to interesting features (Cholesky factorization, etc. see Lecture 3 [9]).

5.11. Levenberg–Marquardt

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^tS is often ill-conditionned (see Lecture 3). One way to decrease significantly the ill-condition feature is to "damp" the system (i.e. using the Tikhonov regularization) using:

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

or better:

$$\left[S^tS + \ell\text{diag}(S^tS)\right]\delta\psi^k = -\nabla j(\psi^k). \tag{55}$$

Note that $\ell \rightarrow 0$ 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.

5.12. Elements of comparison between some presented methods

Some of the presented methods are tested using free software. As usual, the well-known Rosenbrock function is dealts with. In 2–D, this function 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 10 page 20 presents the function. This function presents a long valley where the function gradient is very low.

Next figures present all the evaluations of the function (56). The GSL free library [15] is used for the optimization computation for the simplex, the steepest descent, the conjugate gradient and the BFGS methods. The C program used to generate the minimizing sequence of points for the Rosenbrock function using the BFGS method is given in Appendix 10.2. The other algorithms (CG, Steepest and simplex) are also taken from [15] and slightly change from the source given in subsection 10.2 by only a few lines. Next, the PSO algorithm is the one from [13].

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 [13] 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, respectively 49 and 11 iterations are needed.

6. Functional gradient computation through direct differentiation

6.1. Cost functions, models and derivatives

We recall here that the function to be minimized is the cost function $\mathcal{J}(u)$ 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:

$$j(\psi) := \mathcal{J}(u). \tag{57}$$

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 $\mathcal{S}$ for the state problem. To be concise, one writes down

$$\mathcal{S}(u,\psi) = 0, \tag{58}$$

where we have the mapping $\psi \mapsto u(\psi)$.

Often, the space (and time) is discretized so that the state operator $\mathcal{S}$ is approximated in some matrix formulation. In this case, we have

$$\mathcal{R}(u,\psi) = 0, \tag{59}$$

where $\dim \mathcal{R} = \dim u$. Note that u involved in (58) is continuous while u involved in (59) may be 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 [2] for the technical definitions.

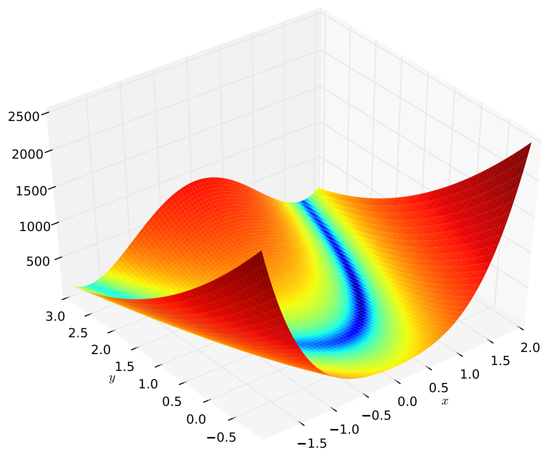

Figure 10. 2–D Rosenbrock function.

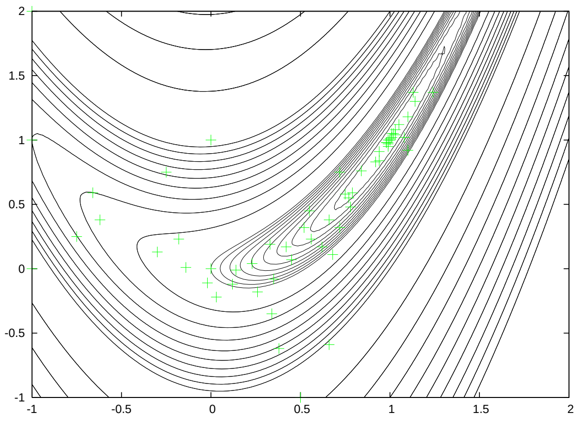

Figure 11. 2–D Rosenbrock function – Simplex algorithm.

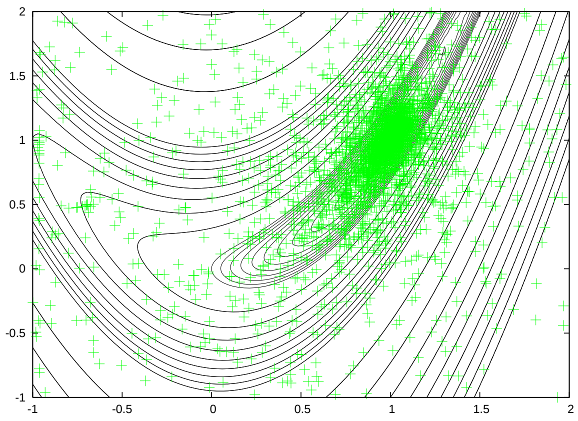

Figure 12. 2–D Rosenbrock function – PSO algorithm.

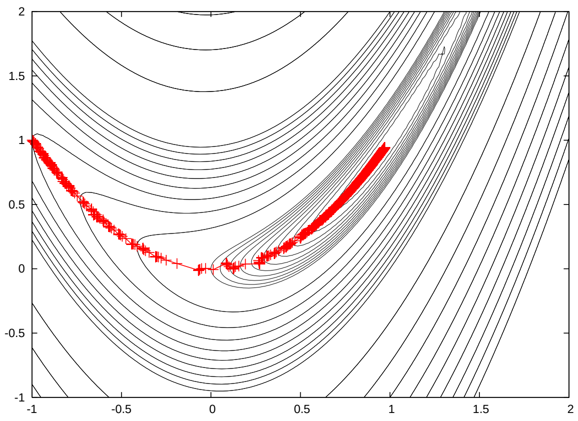

Figure 13. 2–D Rosenbrock function – Steepest descent algorithm.

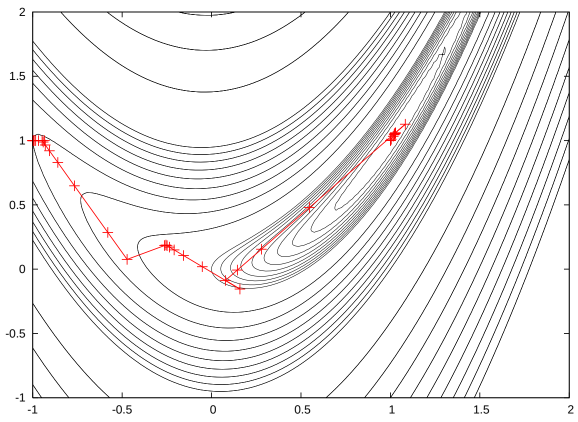

Figure 14. 2–D Rosenbrock function – Conjugate gradients descent algorithm.

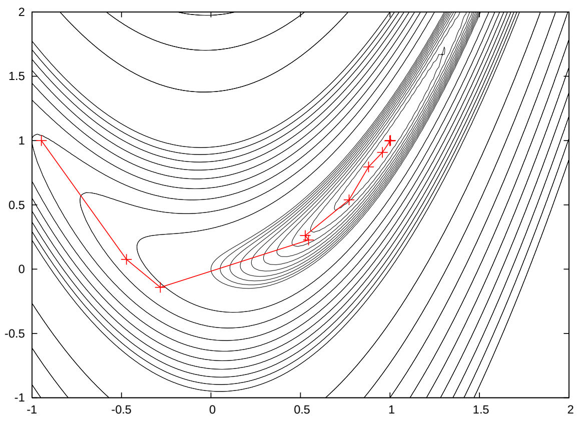

Figure 15. 2–D Rosenbrock function – BFGS descent algorithm.

Definition 6 (Directional derivative) *Let a point ψ and a direction ϕ. 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:*

$$j'(\psi;\phi) := \mathscr{J}'(0) = \lim_{\substack{t\to 0\\ t>0}} \frac{j(\psi + t\phi) - j(\psi)}{t}. \tag{60}$$

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

$$j'(\psi;\delta\psi) = \left(\mathcal{J}'(u), u'(\psi;\delta\psi)\right), \tag{61}$$

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

$$j'(\psi;\delta\psi) = (\nabla j(\psi), \delta\psi). \tag{62}$$

6.2. 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 $\delta\psi = \delta\psi_1, \delta\psi_2, \dots, \delta\psi_{\dim\psi}$. For the i^{th} component, we have:

$$(\nabla j(\psi))_i = (\nabla j(\psi), \delta\psi_i) \approx \frac{j(\psi + \epsilon\delta\psi_i) - j(\psi)}{\epsilon}. \tag{63}$$

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 scalar ε is fixed. The very simple algorithm is described in Algorithm 5.

Algorithm 5: The finite difference algorithm to compute the gradient of the cost function

```
Set the length $\varepsilon$;
At iteration $p$, compute the state $u(\psi^p)$, compute $j(\psi^p)$;
foreach $i = 1, \dots, \dim\psi$ do
    Compute the cost $j(\psi^p + \varepsilon\psi_i\delta\psi_i)$;
    Set the gradient $(\nabla j(\psi))_i \leftarrow \frac{j(\psi^p + \varepsilon\psi_i\delta\psi_i) - j(\psi^p)}{\varepsilon\psi_i}$
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 given by (59), 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, the BFGS, etc.

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 more powerfull quasi-Newton Gauss–Newton or Levenberg–Marquardt methods which both rely on the sensitivities $u'(\psi;\delta\psi_i)$, $i = 1, \dots, \dim\psi$. See Algorithm 6.

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

```
Set the step $\varepsilon$;
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 $u'(\psi;\delta\psi_i) \leftarrow \frac{u(\psi^p+\varepsilon\psi_i\delta\psi_i)-u(\psi^p)}{\varepsilon\psi_i}$;
    Set the gradient $(\nabla j, \delta\psi_i)$ with either $(\mathcal{J}'(u), u'(\psi;\delta\psi_i))$ or as in previous algorithm
    with $\frac{j(\psi^p+\varepsilon\psi_i\delta\psi_i)-j(\psi^p)}{\varepsilon\psi_i}$.
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).
```

6.3. Forward differentiation

The forward differentiation approach consists in computing $u'(\psi;\delta\psi_i)$ differentiating the state equations (continuous or discretized). The state equation being given by (58) for the continuous version, and by (59) for the discrete version, one gets:

$$\mathcal{S}'_u(u,\psi)u' + \mathcal{S}'_\psi(u,\psi)\delta\psi = 0. \tag{64}$$

The discrete version of this relationship can be obtained discretizing (64) or equivalently differentiating (59), giving:

$$\mathcal{R}'_u(u,\psi)u' + \mathcal{R}'_\psi(u,\psi)\delta\psi = 0. \tag{65}$$

As for the previous subsection, the gradient computation needs one computation of (65) per parameter ψ_i, so one needs $\dim\psi$ integrations of (65) to access the full gradient $\nabla j(\psi)$.

As for the finite difference approach, one may use the sensitivities u' and integrate them into the quasi-Newton methods such as the Gauss–newton or Levenberg–Marquardt methods, or only 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 $\mathcal{S}$ and $\mathcal{R}$ are likely to be nonlinear operators, the systems (64) and (65) are linear, thus yielding to much less CPU time. Another singularity is that $\mathcal{R}'_u(u,\psi)$ is the tangent matrix that is to be used anyway for solving the "direct" problem $\mathcal{R}(u,\psi) = 0$. The computation of this linear tangent matrix is most often the task that takes the longer time in solving $\mathcal{R}(u,\psi) = 0$. The optimized procedure is thus the one given in Algorithm 7.

Note that the linear tangent matrix which is to be assembled for the solution of the "direct" forward model is to be re-used for all canonical components $\delta\psi_i$.

7. Functional gradient computation through the adjoint problem

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 7.1) and another one that uses the Lagrange function (subsection 7.2). 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

Algorithm 7: The forward differentiation algorithm to compute the cost gradient and the sensitivities

```
At iteration $p$, solve iteratively $\mathcal{R}(u, \psi^p) = 0$, compute $j(\psi^p)$ and save the linear tangent matrix $\mathcal{R}'_u(u, \psi^p)$;
foreach $i = 1, \dots, \dim \psi$ do
    Solve $\mathcal{R}'_u(u, \psi)u' + \mathcal{R}'_\psi(u, \psi^p)\delta\psi_i = 0$;
    Set $(\nabla j, \delta\psi_i) = (\mathcal{J}'(u), u'(\psi; \delta\psi_i)$;
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).
```

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 of this lecture (section 9).

7.1. Identification method

In this subsection, we derive the method that gives the cost function gradient through the use of an additional adjoint-state problem when the state equation is the continuous version, i.e. (58). This method can be easily transposed for the discrete state version, just substituting $\mathcal{S}$ by $\mathcal{R}$.

From the definition of the functional gradient, one writes the gradient:

$$(\nabla j, \delta\psi) = j'(\psi; \delta\psi) = \left(\mathcal{J}'(u), u'(\psi; \delta\psi)\right). \tag{66}$$

One then introduces a new variable (the adjoint-state variable u^*) such that the gradient equation (66) also satisfies the more–easy–to–compute:

$$j'(\psi; \delta\psi) = \left(\mathcal{S}'_\psi(u, \psi)\delta\psi, u^*\right). \tag{67}$$

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

$$\mathcal{S}'_u(u, \psi)u' + \mathcal{S}'_\psi(u, \psi)\delta\psi = 0 \tag{68}$$

and thus

$$j'(\psi; \delta\psi) = -\left(\mathcal{S}'_u(u, \psi)u', u^*\right). \tag{69}$$

Identifying (66) and (69), we obtain the adjoint-state problem that must satisfy the equality:

$$-\left(\mathcal{S}'_u(u, \psi)u', u^*\right) = \left(\mathcal{J}'(u), u'(\psi; \delta\psi)\right). \tag{70}$$

Next, if the adjoint problem (70) is satisfied (it means that we accessed the adjoint state u^*), then the cost function gradient is very simply given by (67).

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 (70) to:

$$\mathcal{S}^*(u, \psi)u^* + \mathcal{J}'(u) = 0 \tag{71}$$

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

$$\left(\mathcal{S}'_u(u, \psi)u', u^*\right) = \left(S^*(u, \psi)u^*, u'\right) + b \tag{72}$$

where the term b may contain terms (among others boundary terms) usefull for some further simplifications (see Appendix for an example of such simplifications).

Algorithm 8: The adjoint state problem to compute the cost function gradient with integration within an optimization algorithm

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

7.2. Lagrange formulation

The use of a Lagrange formulation means that the state equations are taken as constraints in the optimization problem. In the discrete version of the state, we have $\mathcal{R}(u,\psi)=0$, so that all the equations $\mathcal{R}_i(u,\psi)=0\ \forall i=1,\dots,\dim u$ are taken as constraints.

Let us introduce the Lagrange function [16, 17]:

$$\mathscr{L}(u,u^*,\psi) = \mathcal{J}(u) + (\mathcal{R}(u,\psi),u^*) \tag{73}$$

where the inner product in (73) is $(a,b)=\sum_{i=1}^{\dim u} a_i b_i$. If the state problem is time dependent (say $t\in[0,t_f]$), then the inner product also takes into account time integration: $(a,b) = \int_0^{t_f}\sum_{i=1}^{\dim u} a_i(t)b_i(t)\mathrm{d}t$ as will be seen in the examples dedicated section 9. However, we do not precise here, as in subsection 7.1 the inner product nature.

Note that if a continuous optimization problem is dealt with, the Lagrange function (73) is substituted to $\mathscr{L}(u,u^*,\psi) = \mathcal{J}(u) + (\mathcal{S}(u,\psi),u^*)$, and the inner product can be for instance $(a,b)=\int_0^T\int_\Omega ab\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}t$.

Note also that the lagrangian 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 (59) that maps ψ to u. Moreover, since u is the solution of the modeling problem $\mathcal{R}(u,\psi)=0$, then the lagrangian $\mathscr{L}$ is always equal to the cost function $\mathcal{J}(u)$ 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 (4) 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

$$\exists\,(u,u^*,\psi)\ \mid \mathscr{L}'_u(\cdot)\,\delta w = 0;\ \mathscr{L}'_{u^*}(\cdot)\,\delta w = 0;\ \mathscr{L}'_\psi(\cdot)\,\delta w = 0, \tag{74}$$

for all directions δw taken in appropriate spaces (u', δu^* and $\delta\psi$). First, since the state is satisfied, then $\mathscr{L}'_{u^*} = \mathcal{R}(u,\psi)=0$. Moreover, since we have also $\mathcal{R}'_u(u,\psi)u' + \mathcal{R}'_\psi(u,\psi)\delta\psi = 0$, we get:

$$\mathscr{L}'_\psi(\cdot)\,\delta\psi = \left(\mathcal{R}'_\psi(u,\psi)\delta\psi,u^*\right) = -\left(\mathcal{R}'_u(u,\psi)u',u^*\right) \tag{75}$$

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

$$\left(R'_u(u,\psi)u',u^*\right) + \left(\mathcal{J}'(u),u'(\psi;\delta\psi)\right) = 0. \tag{76}$$

This way we obtain the cost function gradient:

$$\mathscr{L}'_\psi(\cdot)\,\delta\psi = \left(\mathcal{J}'(u), u'(\psi;\delta\psi)\right) = j'(\psi;\delta\psi) = (\nabla j, \delta\psi) \tag{77}$$

The adjoint equation is thus:

$$\mathcal{R}^*(u,\psi)u^* + \mathcal{J}'(u) = 0, \tag{78}$$

and the gradient is given by:

$$\nabla j = \left(\mathcal{R}'_\psi(u,\psi), u^*\right). \tag{79}$$

Summarizing, the minimum of the cost function is to be found at the stationary point of the lagrangian (73). When the adjoint system (78) is satisfied, then the components of the cost function gradient are simply given through the scalar product involved in (79).

7.3. 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.

7.3.1. Case of ODE

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

$$\begin{array}{ll} \mathcal{R}(u,\psi) = \mathscr{C}\dot{u} - \mathscr{B} = 0 & \text{for } t \in \mathcal{I} \\ u = u_0 & \text{for } t = 0, \end{array} \tag{80}$$

where $\mathscr{C}$ may be an inertial scalar or a capacity matrix depending on the related case (with coefficients in $\mathbb{C}$) and $\mathscr{B}$ contains the loadings. Injecting the differentiated time-dependent relationship (80) into the adjoint-state relationship (76) gives:

$$\left(\left[\mathscr{C}\frac{d}{dt} + \mathscr{C}'_u\dot{u} - \mathscr{B}'_u\right]u', u^*\right) + \left(\mathcal{J}'(u), u'(\psi;\delta\psi)\right) = 0 \tag{81}$$

where the inner must be understood as $(a,b) = \int_{\mathcal{I}} \langle a,b\rangle\ \mathrm{d}t = \int_{\mathcal{I}} \sum_{i=0}^{\dim u} a_i b_i\,\mathrm{d}t$.

Transposing the involved matrices (81) becomes:

$$\left(\frac{d}{dt}u', \mathscr{C}^* u^*\right) + \left(u', \left(\mathscr{C}'_u\dot{u} - \mathscr{B}'_u\right)^* u^*\right) + \left(\mathcal{J}'(u), u'(\psi;\delta\psi)\right) = 0 \tag{82}$$

where $\mathscr{A}^*$ is the adjoint of $\mathscr{A}$, i.e. the transposed conjugate. One then integrates by part the first term of (82) to get:

$$-\left(u', \mathscr{C}^*\frac{d}{dt}u^*\right) + \left[\langle u', \mathscr{C}^* u^*\rangle\right]_0^{t_f} + \left(u', \left(\mathscr{C}'_u\dot{u} - \mathscr{B}'_u\right)^* u^*\right) + \left(\mathcal{J}'(u), u'(\psi;\delta\psi)\right) = 0 \tag{83}$$

with the product $\langle a,b\rangle = \sum_{i=1}^{\dim u} a_i b_i$.

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

$$\begin{array}{ll} -\mathscr{C}^*\dot{u}^* + \left(\mathscr{C}'_u\dot{u} - \mathscr{B}'_u\right)^* u^* + \mathcal{J}'(u) = 0 & \text{for } t \in \mathcal{I} \\ u^* = 0 & \text{for } t = t_f\ . \end{array} \tag{84}$$

Some remarks must be given concerning the formulation of the adjoint problem (84):

(i) At first, 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 $u^* = 0$ at time $\tau = 0$, and the time-dependent equation (84) is solved in the forward way in the dual time variable τ which corresponds to the backward way in the primal time variable t.

(ii) The loading component $\mathcal{J}'(u)$ involved in (84) is non-zero only at the selected discrete times where the cost function j is to be integrated.

(iii) If the state problem (80) is linear (say $\mathscr{C}$ and $\mathscr{B}$ do not depend on the solution), then we have the linear tangent operator $\mathcal{R}'_u(u, \psi)$ reduced to $\mathscr{C}\frac{d}{dt}$ only, and the adjoint problem simplifies to:

$$\begin{aligned} -\mathscr{C}^*\dot{u}^* + \mathcal{J}'(u) = 0 \qquad & \text{for } t \in \mathcal{I} \\ u^* = 0 \qquad & \text{for } t = t_f \ . \end{aligned} \tag{85}$$

(iv) The last remark concerns the inherent linear character of the adjoint problem. It is pointed out that even though the direct problem is likely to be nonlinear ($\mathscr{C}$ and/or $\mathscr{B}$ depend(s) on the state u for instance in (80)), the adjoint problem is still linear since the operators do not depend on the adjoint variables. The equivalent remark was given for the forward differentiation method which uses the linear tangent operator.

7.3.2. 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. We deal here with a linear problem (for a nonlinear problem, one must add to the following derivation some elements of the previous derivation –the terms related to the nonlinear behavior–). The state problem thus writes:

$$\mathcal{R}(u, \psi) = \mathscr{K}u - \mathscr{B} = 0 \tag{86}$$

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 (86) into the adjoint (76) gives:

$$\left\langle \mathscr{K}u', u^* \right\rangle + \left(\mathcal{J}'(u), u'(\psi; \delta\psi)\right) = 0 \tag{87}$$

Transposing the diffusive matrix in the first scalar product of (87) gives:

$$\left(u', \mathscr{K}^*u^*\right) + \left(\mathcal{J}'(u), u'(\psi; \delta\psi)\right) = 0 \tag{88}$$

Note that here again, when the matrix $\mathscr{K}$ is symmetric with real coefficients, we have $(\mathscr{K}u, v) = (\mathscr{K}^*v, u) = (\mathscr{K}v, u)$. This appears for instance when discretizing the space through the finite difference method or the finite element method while dealing with purely diffusive systems. Next, since the adjoint problem (88) must be verified for all directional derivatives u', the general adjoint problem becomes:

$$\mathscr{K}^*u^* + \mathcal{J}'(u) = 0. \tag{89}$$

Some remarks must be given concerning the formulation of the adjoint problem (89):

(i) The loading component involved in the space-dependent equation is non-zero only at the selected discrete locations where the cost function j is to be integrated, i.e. at $\boldsymbol{x} = \boldsymbol{x}_m$, with $m = 1 \ldots M$.

(ii) Next, the loading component $\mathcal{J}'(u)$ must be in perfect coherence with the type of discretization (see [18] for detailed explanations when using for instance a finite element discretization scheme).

7.3.3. Case of parabolic PDE

The discretization of the space and time dependent (linear) diffusive model yields to the so-called parabolic problem. It is somehow the union between both just above presented cases. After discretization, the direct problem is:

$$\begin{array}{ll} \mathcal{R}(u,\psi) = \mathscr{C}\dot{u} + \mathscr{K}u - \mathscr{B} = 0 & \text{for } t \in \mathcal{I} \\ u = u_0 & \text{for } t = 0. \end{array} \tag{90}$$

Injecting the differentiated operators involved in (90) into the adjoint (76) gives:

$$\left(\mathscr{C}\frac{d}{dt}u', \psi\right) + \left(\mathscr{K}u', \psi\right) + \left(\mathcal{J}'(u), u'(\psi;\delta\psi)\right) = 0 \tag{91}$$

Transposing all operators through integration by parts gives:

$$-\left(u', \mathscr{C}^*\frac{d}{dt}u^*\right) + \left[\langle u', \mathscr{C}^* u^*\rangle\right]_0^T + \left(u', \mathscr{K}^* u^*\right) + \left(\mathcal{J}'(u), u'(\psi;\delta\psi)\right) = 0 \tag{92}$$

Eventually, the adjoint problem writes:

$$\begin{array}{ll} -\mathscr{C}^*\dot{u}^* + \mathscr{K}^*u^* + \mathcal{J}'(u) = 0 & \text{for } t \in \mathcal{I} \\ \quad u^* = 0 & \text{for } t = t_f. \end{array} \tag{93}$$

7.4. The global optimization algorithm

The general algorithm is given in Algoritm 9. 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 9: The global optimization algorithm

```
(i)   Integrate the cost function value through integration of the forward (maybe nonlinear)
      direct problem;
      Store all state variables to reconstruct the tangent matrix (or store the tangent matrix);
(ii)  Integrate the backward linear adjoint problem, all matrices being possibly stored or
      recomputed from stored state variables
(iii) Compute the cost function gradient;
      Compute the direction of descent
(iv)  Solve the line research algorithm through several integrations of the nonlinear direct
      model.
```

8. 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.

8.1. 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 [5, 3, 4].

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

$$\frac{\|\psi_{k+1}-\psi_k\|}{\|\psi_k-\bar{\psi}\|}\leq \tau, \quad \tau\in(0,1). \tag{94}$$

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*

$$\lim_{k\to\infty}\frac{\|\psi_{k+n}-\psi_k\|}{\|\psi_k-\bar{\psi}\|}=0. \tag{95}$$

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

$$\frac{\|\psi_{k+1}-\psi_k\|}{\|\psi_k-\bar{\psi}\|^2}\leq \tau, \quad \tau>0. \tag{96}$$

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 [3].

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 [3]. However, for the quasi-Newton method, the memory place is proportionnal to n^2.

8.2. Gradient computation cost

Let $\mathcal{R}(u,\psi)=0$ the problem that maps $\psi\mapsto u$, $\mathcal{R}$ being 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 $\mathcal{R}$, the differentiated model and/or the adjoint state model are computed to access the full gradient of the cost function.

(i) Forward finite difference method:
$(\dim\psi+1)$ nonlinear resolution of $\mathcal{R}(u,\psi)=0$.

(ii) Forward differentiation method:
1 nonlinear resolution of $\mathcal{R}(u,\psi)=0$,
$\dim\psi$ linear resolution of $\mathcal{R}'_u(u,\psi)u'+\mathcal{R}'_\psi(u,\psi)\delta\psi=0$.

(iii) Adjoint state method:
1 nonlinear resolution of $\mathcal{R}(u,\psi)=0$,
1 linear resolution of $\mathcal{R}^*(u,\psi)u^*+\mathcal{J}'(u)=0$.

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 computationnaly expensive.

8.3. 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, ...
$u \leftarrow \mathcal{R}(u,\psi)=0$ $j \leftarrow u$ $\nabla j \leftarrow \begin{cases}\text{Forward diff.}\\ \text{or}\\ \text{Adjoint state}\end{cases}$	$u \leftarrow \mathcal{R}(u,\psi)=0$ $j \leftarrow u$ $\nabla j \leftarrow \begin{cases}\text{Forward diff.}\\ \text{or}\\ \text{Adjoint state}\end{cases}$ $\nabla^2 j$ (complicated)	$u \leftarrow \mathcal{R}(u,\psi)=0$ $j \leftarrow u$ $\nabla j \leftarrow S^tS \leftarrow S \leftarrow u'$ (Forward diff.)

9. Example : estimation of convection heat transfer during quenching

In this section, we consider an application of the Iterative Regularization Method to a nonlinear transient heat transfer inverse problem arising in thermal treatment of metals, in particular in Jominy end quench tests. Quenching is one of the most critical operations in the heat treatment of many metallic parts, affecting in general both mechanical and structural properties. The analyzed inverse problem consists in estimating a space and time dependent heat transfer coefficient at the quenching surface. Details of these investigations can be found in [19, 20, 21].

9.1. Introduction

The Jominy end quench test is a standard test (NF A 04-303) used to characterize the hardenability of steels [7,8]. A steel cylinder of diameter d =25 mm is heated within the austenite domain during a preset time and then cooled down by a water jet sprayed on its lower end. The numerical simulation of rapid metallurgical transformations occurred during quenching requires a good knowledge of the heat transfer conditions at the boundaries of the considered domain. The direct measurement of these conditions is impossible and it is necessary to solve an inverse heat transfer problem to estimate them. The experimental setup built according to the standard gives the following characteristics (Figure 1):

- Induction heating in an enclosure filled by an inert gas (argon);
- Computer-aided control of various actions (heating, cylinder handling, quenching) which leads to a good enough reproducibility of the tests.

In this work, two materials are analyzed. The first one is nickel. There is no metallurgical phase changes in this material in the analyzed temperature domain. The second is the 16MND5 steel sample (considered as an equivalent of ASTM A508 Cl. 3 in USA) for which the coupled thermo-metallurgical phenomenon has to be considered.

9.2. Inverse problem

The main difficulty of the problem is the simulation of the coupled thermo-metallurgical phenomena when cooling down a fully austenitized steel specimen with a high nonlinearity of the thermal conductivity during the phase transformations (austenite- martensite). The quenching tests are axi-symetric. The modeling equation in $\Omega = [O, r_{\max}] \times [0, z_{\max}]$ is

$$C\frac{\partial T}{\partial t} - \nabla \cdot (\lambda \nabla T) = S_{pc} \tag{97}$$

where $C(T)$ is the equivalent volumetric heat capacity, $\lambda(T)$ is the equivalent thermal conductivity, and $S_{pc}\,(T)$ is the source term determining the metallurgical phase change thermal

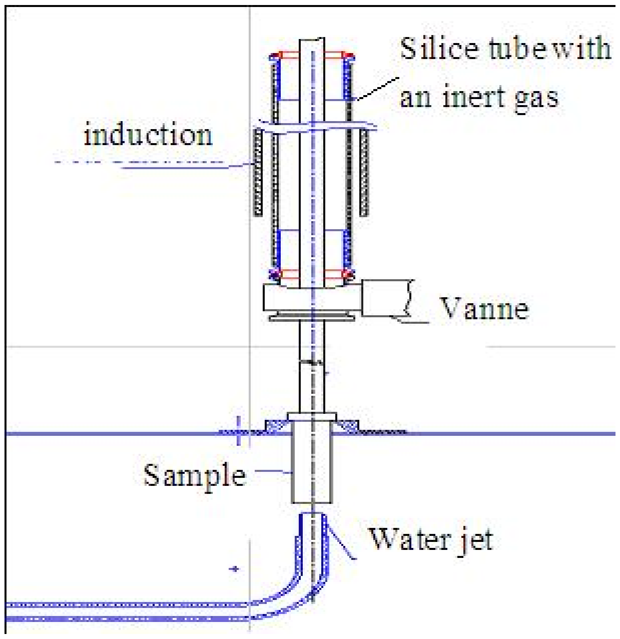

Figure 16. Sketch of experimental setup

effects in the specimen during cooling. Note the the diffusion operator in a cylindrical basis is written as $\nabla \cdot (\lambda \nabla T) = \frac{1}{r}\frac{\partial}{\partial r}\left(r\lambda \frac{\partial T}{\partial r}\right) + \frac{\partial}{\partial z}\left(\lambda \frac{\partial T}{\partial z}\right)$. We consider the following initial condition

$$T = T_0 \tag{98}$$

and the following boundary condition

$$\begin{array}{ll} -\lambda \nabla T \cdot \boldsymbol{n} = h\,(T - T_\infty) & \text{for } z = 0 \\ \nabla T \cdot \boldsymbol{n} = 0 & \text{for } z = z_{\max} \\ \nabla T \cdot \boldsymbol{n} = 0 & \text{for } r = 0 \\ -\lambda \nabla T \cdot \boldsymbol{n} = \varepsilon\sigma\left(T^4 - T_\infty^4\right) & \text{for } r = r_{\max} \end{array} \tag{99}$$

In (97) and (99), the thermophysical characteristics $C(T)$ and $\lambda(T)$ as well as the source term $S_{pc}(T)$ are computed by using the metallurgical kinetic model of the type Koistinen–Marburger [22] and Leblond–Devaux [23] of the form

$$\frac{dP}{dt} = \frac{P_{eq} - P}{\tau} F\left(\frac{dT}{dt}\right) \tag{100}$$

and

$$P = P_{\max}\left(1 - \exp\left(-b(M_s - T)\right)\right) \tag{101}$$

where P is the proportion of a metallurgical phase, P_{eq} is the concentration of this phase in the equilibrium state, τ is the delay time, $F(\frac{dT}{dt})$ is the coefficient characterizing the transformation speed of the phase, $P_{\max}$ is the maximum proportion before the transformation, b is the coefficient determining the final temperature of the phase transformation, M_s is the beginning temperature of the transformation.

Figure 17 shows a Continuous Cooling Temperature (C.C.T.) diagram which is used for the simulation of the metallurgical transformations. This diagram is constructed with a lot of

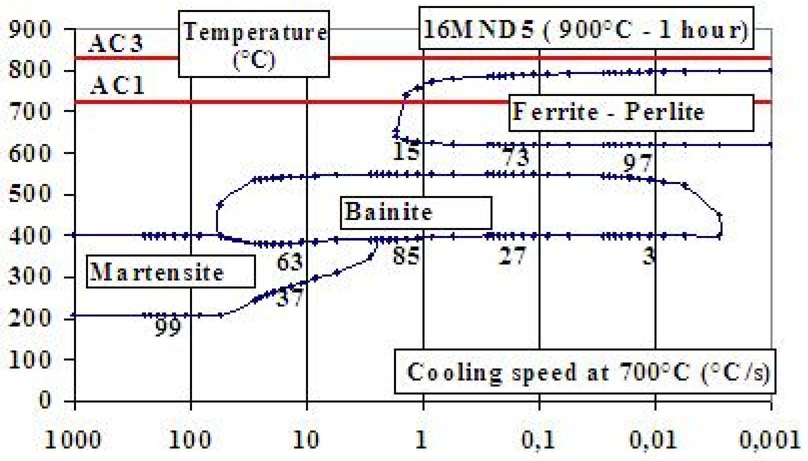

Figure 17. CCT diagram

dilatometric experiments. Each experiment is realised with a steel specimen initially heated at $T_0 = 880$ °C during a preset time (30 mn in this case). During the cooling, the parameter which influences the transformation is the cooling speed. In function of this speed, the transformation begins and stops at different temperatures (austenite-ferrite: 800 °C> T >600 °C, austenite-bainite: 550 °C> T >380 °C, and austenite-martensite: 400 °C> T >200 °C.)

For the simulation of the transformations austenite-ferrite and austenite-bainite, we use the equation (100) and for the transformation austenite-martensite the equation (101). All parameters of these equations are defined with the CCT diagram [24].

During the metallurgical transformation, a phase change heat is computed with the use of the phase enthalpy: $L_{\gamma\alpha} = \rho_\gamma H_\gamma - \rho_\alpha H_\alpha$ and by considering only two metallurgical phases: γ (austenite) and α (ferrite, bainite or martensite). The enthalpies of the phase γ and α are approximated with a good accuracy by polynomials between 100 °C and 1450 °C [24].

A finite-difference method is used to simulate numerically the Jominy tests. For each node of the finite-difference grid, the temperature and the cooling speed are computed. With these two parameters and the equation (100) or (101), we can calculate the proportions of the metallurgical phases

Then, the thermophysical characteristics $C(T)$ and $\lambda(T)$ are computed at all nodes by using the mixture law as functions of temperature as follows

$$C(T) = P_\gamma \rho_\gamma c_\gamma(T) - P_\alpha \rho_\alpha c_\alpha(T) \tag{102}$$

$$\lambda(T) = P_\gamma \lambda_\gamma(T) - P_\alpha \lambda_\alpha(T) \tag{103}$$

The source term is computed as

$$S_{pc}(T) = \Delta P_{\gamma\alpha} L_{\gamma\alpha} \tag{104}$$

where $\Delta P_{\gamma\alpha}$ is the proportion of the transformed phase.

To obtain the information on the temperature distribution needed to solve the inverse problem, a set of thermocouples is installed in the specimen near the quenching surface at N discrete points with coordinates $\boldsymbol{x}_n = (r_n, z_n)$, $n = 1, \ldots, N$ (measurement scheme) and the distance between 0.5 mm and 12 mm from the quenching surface. The measured temperature evolutions can be presented as

$$T(r_n, z_n, t) = f_n(t), \quad n = 1, \ldots, N. \tag{105}$$

The nonlinear inverse problem under analysis consists in estimating a convection heat transfer coefficient $h(r,t)$ from the conditions (97) to (105).

9.3. Numerical algorithm

The estimation of the heat transfer coefficient in view of the direct problem (equations (97) to (100)) can be formulated under the variational form which implies the residual functional minimization.

$$j(h) := \mathcal{J}(T) = \int_0^{t_f} \sum_{j=1}^{n} (T(r_j, z_j, t) - Y(r_j, z_j, t))^2 dt \tag{106}$$

where $T(r_j, z_j, t)$ and $Y(r_j, z_j, t)$ represent respectively the estimated and measured temperatures at N various points of the material. The inverse problem consists in minimizing this residual functional under constraints given by the equations of the direct system (equations (97) to (100)). The minimization is carried out by using the conjugate gradient method. The function $h(r,t)$ is considered here as an element of the Hilbert space $L_2(\Gamma)$, Γ being the boundary at $z = 0$, and the new functions are obtained after each iteration as follows:

$$h^{p+1} = h^p + \alpha^p d^p \tag{107}$$

where p is the iteration index, α^p is the descent parameter, h^{p+1} the unknown vector to be estimated and d^p the vector of descent direction given by:

$$d^p = -\nabla j^p + \beta^p d^{p-1} \tag{108}$$

where we used the Polak and Ribière's method recalled here:

$$\beta^p = \frac{\left(\nabla j(\psi^p), \nabla j(\psi^p) - \nabla j(\psi^{p-1})\right)}{\left(\nabla j(\psi^{p-1}), \nabla j(\psi^{p-1})\right)}, \tag{109}$$

with the inner product being held in $L_2(\Gamma)$. In the absence of noise, the iteration procedure is carried on until the following stopping criterion is verified:

$$\left| \frac{h^{p+1} - h^p}{h^p} \right| \leq \epsilon_1 \tag{110}$$

The cost function gradient is obtained for all values of r and t by the following analytical relationship (see Appendix 10.1):

$$\nabla j(h(r,t)) = T^*(r,0,t) \left[T(r,0,t) - T_\infty\right] \tag{111}$$

where $T^*(r,0,t)$ is the solution of the adjoint problem achieved during the integration of the Lagrange functional and of the variation problem $\theta\,(r,z,t)$:

$$\frac{\partial (C\theta)}{\partial t} - \nabla \cdot (\nabla(\lambda\theta)) = \Delta P_{\gamma\alpha} \frac{\partial}{\partial T} (\rho_\gamma H_\gamma - \rho_\alpha H_\alpha)\theta \tag{112}$$

in the domain Ω, with the initial condition

$$\theta(r,z,0) = 0 \tag{113}$$

and the following boundary conditions:

$$\begin{array}{ll} \nabla(\lambda\theta) \cdot \boldsymbol{n} = h\theta + \delta h (T - T_\infty) & \text{for } z = 0 \\ \nabla\theta \cdot \boldsymbol{n} = 0 & \text{for } z = z_{\max} \\ \nabla\theta \cdot \boldsymbol{n} = 0 & \text{for } r = 0 \\ \nabla(\lambda\theta) \cdot \boldsymbol{n} = 4\varepsilon\sigma T^3 \theta & \text{for } r = r_{\max} \end{array} \tag{114}$$

For the adjoint problem (see Appendix 10.1 page 38 for the technical "sketch of the proof"), we obtain in Ω:

$$-C\frac{\partial T^*}{\partial t}-\nabla\cdot(\lambda\nabla T^*) = \Delta P_{\gamma\alpha}\frac{\partial}{\partial T}(\rho_\gamma H_\gamma-\rho_\alpha H_\alpha)T^*+\sum_{j=1}^{n}(T(r_j,z_j,t)-Y(r_j,z_j,t))\delta(r-r_j)\delta(z-z_j) \tag{115}$$

with the condition at final time t_f:

$$T^*(r,z,t_f)=0 \tag{116}$$

and the conditions on the boundaries:

$$\begin{array}{ll} -\lambda\nabla T^*\cdot\boldsymbol{n}=hT^* & \text{for } z=0 \\ \nabla T^*\cdot\boldsymbol{n}=0 & \text{for } z=z_{\max} \\ \nabla T^*\cdot\boldsymbol{n}=0 & \text{for } r=0 \\ -\lambda\nabla T^*\cdot\boldsymbol{n}=4\varepsilon\sigma T^3T^* & \text{for } r=r_{\max} \end{array} \tag{117}$$

Next, rather than performing a line-search, assuming that the cost function is close to be quadratic, we use for the step α^p the relationship:

$$\alpha^n=-\frac{\sum_{j=1}^{N}\int_0^{t_f}\left[T\left(r_j,z_j,t\right)-Y\left(r_j,z_j,t\right)\right]\theta\left(r_j,z_j,t\right)\,\mathrm{d}t}{\sum_{j=1}^{N}\int_0^{t_f}\theta\left(r_j,z_j,t\right)^2\,\mathrm{d}t} \tag{118}$$

where $\theta\left(r_j,z_j,t\right)$ is the solution of the variation problem computed on the direction δh equal to the direction of descent, and N is the number of measurement points.

The three systems are solved numerically using a finite difference method and an implicit scheme. The estimation procedure consists in following Algorithm 10.

Algorithm 10: Estimation procedure for the estimation of the heat transfer coefficient

```
(i) Defining the initial values of de $h^0(r,t)$
(ii) Solving the direct system
(iii) Calcultating the cost function $j(h)$
(iv) Verifying the convergence criterion
(v) Solving the adjoint system
(vi) Calculating the gradient vector
(vii) Calculating the vector of descent direction
(viii) Solving the variation problem
(ix) Calculating the descent parameter
(x) Incrementing the vector $h^{p+1}(r,t)$
(xi) And go back to step (ii)
```

The inverse problems are ill-posed and numerical solutions depend on the fluctuations occurring in the measurements. Small fluctuations in the measurements can generate big errors in the solution. We use the iterative regularization method in which the regularizing condition is the residual criterion:

$$j\left(h^{p^*}\right)\approx\delta^2 \tag{119}$$

where p^* is the index of the last iteration at which the condition $j\,(h) \leq \delta^2$ is verified for the first time, δ^2 is the total (integrated) measurement error defined as:

$$\delta^2 = \int_0^{t_f} \sum_{j=1}^{N} \sigma^2(r_j, z_j, t)\,\mathrm{d}t \qquad (120)$$

$\sigma^2(r_j, z_j, t)$ being the root-mean-square error of the temperature measurements obtained by smoothing the measured temperature histories. The number n^* is the regularization parameter of the method.

9.4. Numerical experiments

In order to verify the method, two materials were analyzed. One was nickel, without metallurgical transformation and another one was 16MND5 steel. We can find the results of the analyze in the paper [21]. Here we want to show the theoretical analyzis of the Iterative Regularization Method based on the conjugate gradient method. For the analyzis, we work with a heat transfer coefficient which varies along a dome (Figure 18). We want to underline few problems of the IRM method.

9.4.1. Analyze without noise

At first, in the adjoint problem at the final time, the adjoint variable is equal to zero. So the residual functional and the descent direction are equal to zero and the estimation cannot be obtained at this final time. The initial "guess" value is $h(r,t) = 1000$ W/m^2 (see Figure 18). The obtained coefficient is presented in Figure 19.

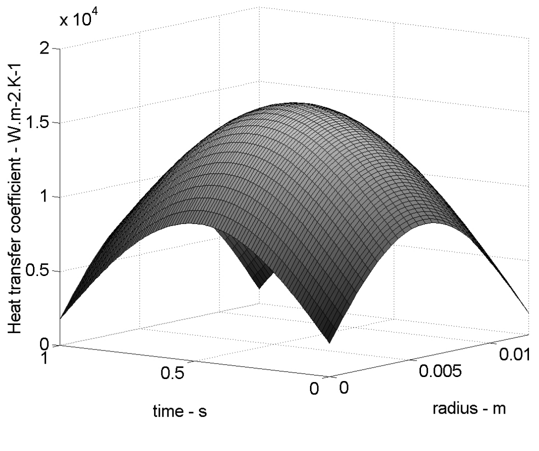

Figure 18. Heat transfer coefficient.

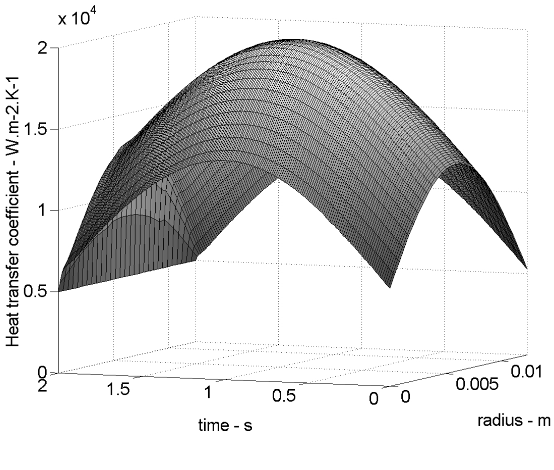

Figure 19. Estimated heat transfer coefficient.

The first analysis of the different estimates confirms that the errors are all the more big since we are moving away from the surface ($z = 0$) and especially when approaching $t = t_f$. Increasing the estimation domain by 25 to 30 % beyond the effective time is advisable. Figures 20 and 21 show respectively the results in the case of the dome for $t_f = t_f + 30$ %, and then truncated to t_f.

9.4.2. Analyzis with noise

In a second phase, we verify that, even in the presence of noise, the convergence of the code toward an estimation is acceptable. The case of the dome only, for which a noise has been

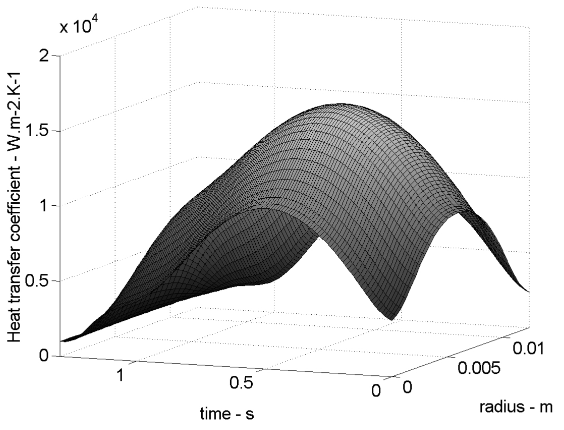

Figure 20. Estimated heat transfer coefficient with time increase.

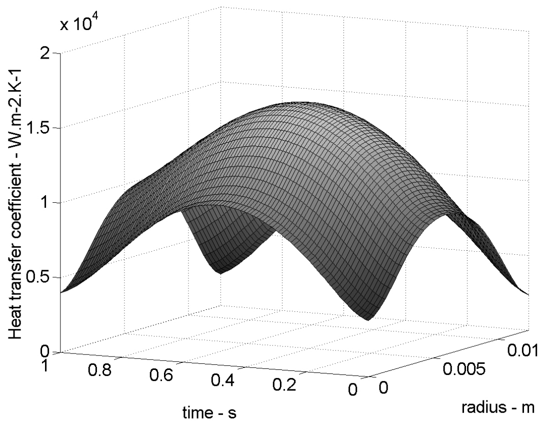

Figure 21. Estimated heat transfer coefficientwith time increase and trunction.

added to the theoretical thermal cycles, is presented here. The disturbance is defined by the relationship: $B = \omega \times T_{\max} \times \Delta_{\max}$ where ω is a random number generator in [-1, +1], $T_{\max}$ is the maximum value of the surface temperature and $\Delta_{\max}$ is the magnitude of the disturbance. For these numerical tests, we consider $T_{\max} = 880$ °C and $\Delta_{\max} = 5$ %, and $Y_j(r_j, z_j, t) \leftarrow Y_j(r_j, z_j, t) + \omega \times T_{\max} \times \Delta_{\max}$.

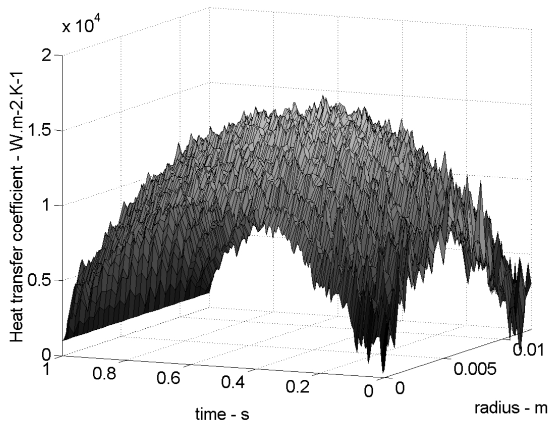

Figure 22. Values estimated in the case of the dome with noisy data.

Figure 22 shows the results achieved in the case of the dome in $z = 0$ mm and for $t_f = 1$s. When the criterion $j(h)$ is equal to δ^2 (iteration 8) the best result is obtained. If the estimation procedure goes on, the criterion continues to decrease. On the other hand, noise is added on the estimated values, see Figure 23.

The last analyzed point concerns the number of thermocouples. In fact, if we want to estimate ten values in the radius direction, we must to have ten thermocouples. So if we don't have these ten thermocouples, we must construct an estimation with a parametrization. For example, we

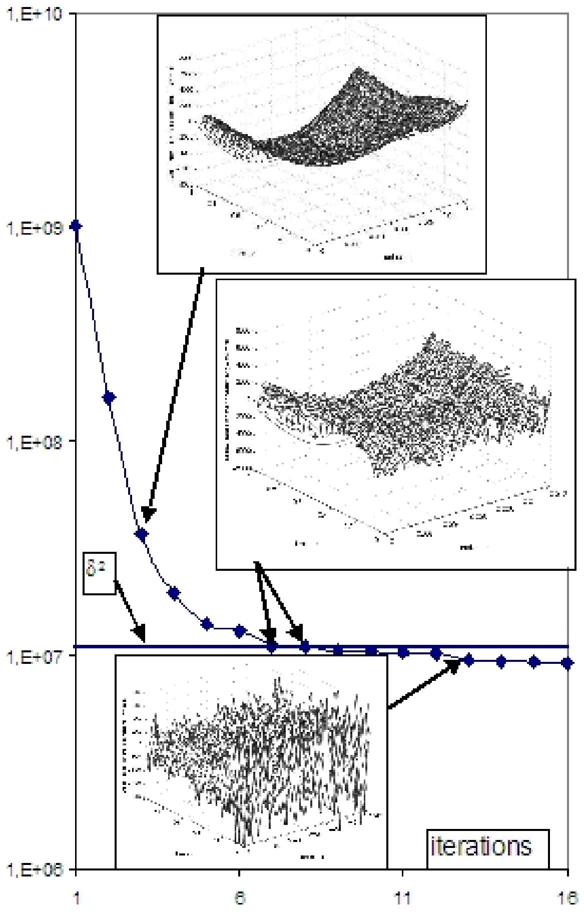

Figure 23.

use a parametric representation of the heat transfer coefficient:

$$h(r,t) = \sum_{i=1}^{M_1} \sum_{j=1}^{M_2} p_{ij} \eta_{ij}(r,t) \tag{121}$$

where p_{ij}, $i = 1, \ldots, M_1$, $j = 1, \ldots, M_2$ are approximation parameters and $\eta_{ij}(r,t)$ are given basis cubic B-splines. The goal in this case is to estimate the p_{ij} parameters with Algorithm 11.

Algorithm 11: Estimation procedure for the estimation of the parametrized heat transfer coefficient

```
(i) Defining the initial values of de $p^0(r,t)$
(ii) Calculating the values of $h^0(r,t)$
(iii) Solving the direct system
(iv) Calcultating the cost function $j(h)$
(v) Verifying the convergence criterion
(vi) Solving the adjoint system
(vii) Calculating the gradient vector
(viii) Calculating the vector of descent direction
(ix) Solving the variation problem
(x) Calculating the descent parameter $\gamma^n$
(xi) Incrementing the vector $p^{p+1}(r,t)$
(xii) And go back to step (iii)
```

References

[1] E. Ruffio, D. Saury, D. Petit, and M. Girault. Tutorial 2: Zero-order optimization algorithms. Eurotherm School METTI 2011 : Thermal measurements and inverse techniques, Roscoff, 2011.

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

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

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

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

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

[7] 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.

[8] J.-L. Battaglia. Lecture 8: Experimental modelling through identification of low order models. Eurotherm School METTI 2011 : Thermal measurements and inverse techniques, Roscoff, 2011.

[9] F. Rigollet. Lecture : Title??? Eurotherm School METTI 2011 : Thermal measurements and inverse techniques, Roscoff, 2011.

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

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

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

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

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

[15] M. Galassi. *GNU Scientific Library Reference Manual (http://www.gnu.org/software/gsl/).*

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

[17] 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.

[18] Y. Favennec, V. Labbé, and F. Bay. Induction heating processes optimization – a general optimal control approach. *J. Comput. Phys.*, 187:68–94, 2003.

[19] P. Le Masson, T. Loulou, E. Artioukhine, P. Rogeon, P. Carron, and J.-J. Quemener. A numerical study for the estimation of a convection heat transfer coefficient during a metallurgical jominy end-quench test. *International Journal of Thermal Sciences*, 41(6):517–527, 2002.

[20] P. Le Masson, T. Loulou, and E. Artioukhine. Comparison between methods for estimating the convection heat transfer coefficient during a metallurgical jominy end-quench test. In E papers Publishing House Ltd., editor, *Proceedings of the 4th International Conference on Inverse Problems in Engineering: Theory and Practice*, pages 171–178, May 2002.

[21] P. Le Masson, T. Loulou, and E. Artiokhine. Experimental and numerical estimation of a two-dimensional convection heat transfer coefficient during metallurgical jominy end quench test. In *Proc. of the 4th Int. Conf. on Inverse Problems in Engineering: Theory and Practice*, pages 179–186, May 2002.

[22] D.P. Koistinen and R.E. Marburger. A general equation prescribing the extent of austenite-martensite transformation in pure iron-carbon alloys and plain carbon steels. *Acta. Metall.*, 7(59), 1959.

[23] J. Leblond and J. Devaux. A new kinetic model for anisothermal metallurgical transformations in steels including effect of austenite grain size. *Acta Metall.*, 32(1):137–146, 1984.

[24] M. Costantini. *Simulation numérique du soudage par faisceau d'électrons - contribution au développement d'un modèle prédictif de l'apport d'énergie.* PhD thesis, Université Paris 6, 1985.

[25] Y. Jarny, M.N. Ozisik, and J.P. Bardon. A general optimization method using adjoint equation for solving multidimensional inverse heat conduction. *International Journal of Heat and Mass Transfer*, 34:2911–2919, 1991.

10. Appendix

10.1. Sketch of the demonstration of the adjoint problem for the quenching problem

We recall here the equations of the "direct" modeling that solves $\mathcal{S}(T,h)=0$:

$$\begin{cases} C\frac{\partial T}{\partial t} - \nabla\cdot(\lambda\nabla T) = S_{pc} & \boldsymbol{x}\in\Omega,\ t\in\mathcal{I} \\ T = T_0 & \boldsymbol{x}\in\Omega,\ t=0 \\ \lambda\nabla T\cdot\boldsymbol{n} = -h\left(T-T_\infty\right) & \boldsymbol{x}\in\partial\Omega_1,\ t\in\mathcal{I} \\ \nabla T\cdot\boldsymbol{n} = 0 & \boldsymbol{x}\in\partial\Omega_2\cup\Omega_3,\ t\in\mathcal{I} \\ \lambda\nabla T\cdot\boldsymbol{n} = -\varepsilon\sigma\left(T^4-T_\infty^4\right) & \boldsymbol{x}\in\partial\Omega_4,\ t\in\mathcal{I} \end{cases} \tag{122}$$

where $\Omega=[O,r_{\max}]\times[0,z_{\max}]$, $\mathcal{I}=[0,t_f]$, $\partial\Omega_1=[O,r_{\max}]\times(z=0)$, $\partial\Omega_2=[O,r_{\max}]\times(z=z_{\max})$, $\partial\Omega_3=(r=0)\times[0,z_{\max}]$, and $\partial\Omega_4=(r=r_{\max})\times[0,z_{\max}]$.

The "sensitivity" problem (i.e. the direction derivative of the state T in the direction δh here denoted $\theta(h;\delta h)$ rather than $T'(h;\delta h)$) is defined by:

$$\begin{cases} \frac{\partial(C\theta)}{\partial t} - \nabla\cdot(\nabla(\lambda\theta)) - \Delta P_{\gamma\alpha}\frac{\partial}{\partial T}(\rho_\gamma H_\gamma - \rho_\alpha H_\alpha)\theta = 0 & \boldsymbol{x}\in\Omega,\ t\in\mathcal{I} \\ \theta = 0 & \boldsymbol{x}\in\Omega,\ t=0 \\ \nabla(\lambda\theta)\cdot\boldsymbol{n} + h\theta + \delta h(T-T_\infty) = 0 & \boldsymbol{x}\in\partial\Omega_1,\ t\in\mathcal{I} \\ \nabla\theta\cdot\boldsymbol{n} = 0 & \boldsymbol{x}\in\partial\Omega_2\cup\Omega_3,\ t\in\mathcal{I} \\ \nabla(\lambda\theta)\cdot\boldsymbol{n} + 4\varepsilon\sigma T^3\theta = 0 & \boldsymbol{x}\in\partial\Omega_4,\ t\in\mathcal{I} \end{cases} \tag{123}$$

The Lagrange function is formely defined as:

$$\begin{aligned} \mathscr{L}(T,\{T^*,\eta,\gamma,\xi,\varpi\},h) = \ & \mathcal{J}(T) + \left(C\frac{\partial T}{\partial t} - \nabla\cdot(\lambda\nabla T) - S_{pc}, T^*\right)_{L_2(0,T;L_2(\Omega))} \\ & + (T-T_0,\eta)_{L_2(\Omega)}\,(t=0) \\ & + \left(\lambda\nabla T\cdot\boldsymbol{n} + h\left(T-T_\infty\right),\gamma\right)_{L_2(0,T;L_2(\partial\Omega_1))} \\ & + (\nabla T\cdot\boldsymbol{n},\xi)_{L_2(0,T;L_2(\partial\Omega_2\cup\partial\Omega_3))} \\ & + \left(\lambda\nabla T\cdot\boldsymbol{n} + \varepsilon\sigma\left(T^4-T_\infty^4\right),\varpi\right)_{L_2(0,T;L_2(\partial\Omega_4))} \end{aligned} \tag{124}$$

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

$$\begin{aligned} (\mathscr{L}'_h(\cdot),\delta h) = \ & (\mathcal{J}'(T),\theta)_{L_2(0,T;L_2(\Omega))} \\ & + \left(\frac{\partial(C\theta)}{\partial t} - \nabla\cdot(\nabla(\lambda\theta)) - \Delta P_{\gamma\alpha}\frac{\partial}{\partial T}(\rho_\gamma H_\gamma - \rho_\alpha H_\alpha)\theta, T^*\right)_{L_2(0,T;L_2(\Omega))} \\ & + (\theta,\eta)_{L_2(\Omega)}\,(t=0) \\ & + (\nabla(\lambda\theta)\cdot\boldsymbol{n} + h\theta + \delta h(T-T_\infty),\gamma)_{L_2(0,T;L_2(\partial\Omega_1))} \\ & + (\nabla\theta\cdot\boldsymbol{n},\xi)_{L_2(0,T;L_2(\partial\Omega_2\cup\partial\Omega_3))} \\ & + \left(\nabla(\lambda\theta)\cdot\boldsymbol{n} + 4\varepsilon\sigma T^3\theta,\varpi\right)_{L_2(0,T;L_2(\partial\Omega_4))} \end{aligned} \tag{125}$$

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

$$\begin{aligned}\left(\tfrac{\partial(C\theta)}{\partial t}, T^*\right)_{L_2(0,T;L_2(\Omega))} &= \left(\theta, -C\tfrac{\partial T^*}{\partial t}\right)_{L_2(0,T;L_2(\Omega))} + (C\theta, T^*)_{L_2(\Omega)}\,(t=t_f)\\ (\Delta(\lambda\theta), T^*)_{L_2(0,T;L_2(\Omega))} &= (\lambda\Delta T^*, \theta)_{L_2(0,T;L_2(\Omega))}\\ &\quad + (\lambda T^*, \nabla\theta\cdot\boldsymbol{n})_{L_2(0,T;L_2(\partial\Omega))}\\ &\quad + (\nabla\lambda\cdot\boldsymbol{n}T^*, \theta)_{L_2(0,T;L_2(\partial\Omega))}\\ &\quad - (\lambda\nabla T^*\cdot\boldsymbol{n}, \theta)_{L_2(0,T;L_2(\partial\Omega))}\end{aligned} \tag{126}$$

We bring together similar terms to get:

$$\left(\mathscr{L}'_h(T, \{T^*, \eta, \gamma, \xi, \varpi\}, h), \delta h\right) = \text{(I)} + \text{(II)} + \text{(III)} + \text{(IV)} + \text{(V)} \tag{127}$$

where

$$\text{(I)} := \left(\mathcal{J}'(T), \theta\right)_{L_2(0,T;L_2(\Omega))} \tag{128}$$

$$\text{(II)} := \left(-C\frac{\partial T^*}{\partial t} - \lambda\Delta T^* - \Delta P_{\gamma\alpha}\frac{\partial}{\partial T}(\rho_\gamma H_\gamma - \rho_\alpha H_\alpha)T^*, \theta\right)_{L_2(0,T;L_2(\Omega))} \tag{129}$$

$$\begin{aligned}\text{(III)} := &(\nabla\lambda\cdot\boldsymbol{n}T^* - \lambda\nabla T^*\cdot\boldsymbol{n}, \theta)_{L_2(0,T;L_2(\partial\Omega))} + (\nabla\lambda\cdot\boldsymbol{n}\gamma + h\gamma, \theta)_{L_2(0,T;L_2(\partial\Omega_1))}\\ &+ (\nabla\lambda\cdot\boldsymbol{n}\varpi, \theta)_{L_2(0,T;L_2(\partial\Omega_4))} + \left(4\epsilon\sigma T^3\varpi, \theta\right)_{L_2(0,T;L_2(\partial\Omega_4))}\end{aligned} \tag{130}$$

$$\begin{aligned}\text{(IV)} := &(\lambda T^*, \nabla\theta\cdot\boldsymbol{n})_{L_2(0,T;L_2(\partial\Omega))} + (\lambda\gamma, \nabla\theta\cdot\boldsymbol{n})_{L_2(0,T;L_2(\partial\Omega_1))}\\ &+ (\xi, \nabla\theta\cdot\boldsymbol{n})_{L_2(0,T;L_2(\partial\Omega_2\cup\partial\Omega_3))} + (\lambda\varpi, \nabla\theta\cdot\boldsymbol{n})_{L_2(0,T;L_2(\partial\Omega_4))}\end{aligned} \tag{131}$$

$$\text{(V)} := (CT^*, \theta)_{L_2(\Omega)}\,(t=t_f) \tag{132}$$

We choose:

$$(CT^*, \theta)_{L_2(\Omega)}\,(t=t_f) \Longrightarrow T^*(t_f) = 0 \tag{133}$$

$$(\lambda\varpi + \lambda T^*, \nabla\theta\cdot\boldsymbol{n})_{L_2(0,T;L_2(\partial\Omega_4))} = 0 \Longrightarrow \varpi + T^*|_{\partial\Omega_4} = 0 \tag{134}$$

$$\begin{aligned}&\left(\nabla\lambda\cdot\boldsymbol{n}\varpi + 4\epsilon\sigma T^3\varpi + \nabla\lambda\cdot\boldsymbol{n}T^* - \lambda\nabla T^*\cdot\boldsymbol{n}, \theta\right)_{L_2(0,T;L_2(\partial\Omega_4)} = 0\\ &\qquad\qquad \Longrightarrow -\lambda\nabla T^*\cdot\boldsymbol{n} - 4\epsilon\sigma T^3T^*|_{\partial\Omega_4} = 0\end{aligned} \tag{135}$$

$$(\nabla\lambda\cdot\boldsymbol{n}T^* + \lambda\nabla T^*\cdot\boldsymbol{n}, \theta)_{L_2(0,T;L_2(\partial\Omega_2\cup\partial\Omega_3))} = 0 \Longrightarrow \nabla T^*\cdot\boldsymbol{n}|_{\partial\Omega_2\cup\partial\Omega_3} = 0 \tag{136}$$

$$(\xi + \lambda T^*, \nabla\theta\cdot\boldsymbol{n})_{L_2(0,T;L_2(\partial\Omega_2\cup\partial\Omega_3))} = 0 \Longrightarrow \xi + \lambda T^*|_{\partial\Omega_2\cup\partial\Omega_3} = 0 \tag{137}$$

$$(\lambda\gamma + \lambda T^*, \nabla\theta\cdot\boldsymbol{n})_{L_2(0,T;L_2(\partial\Omega_1))} \Longrightarrow \gamma + T^*|_{\partial\Omega_1} = 0 \tag{138}$$

$$(\gamma\nabla\lambda\cdot\boldsymbol{n} + h\gamma + \nabla\lambda\cdot\boldsymbol{n}T^* - \lambda\nabla T^*\cdot\boldsymbol{n}, \theta)_{L_2(0,T;L_2(\partial\Omega_1))} = 0 \Longrightarrow -\lambda\nabla T^*\cdot\boldsymbol{n} - hT^*|_{\partial\Omega_1} = 0 \tag{139}$$

Note that we used $\nabla\lambda\cdot\boldsymbol{n} = \lambda'(T)\nabla T\cdot\boldsymbol{n} = 0$ in order to perform the simplification in (136). Note also that the relationship (137) does not bring any "usable" relationship. Next, note that the relationship (138) has been injected into (139) for the simplification.

The adjoint problem eventually is:

$$\begin{cases} -C\frac{\partial T^*}{\partial t} - \lambda\Delta T^* - \Delta P_{\gamma\alpha}\frac{\partial}{\partial T}(\rho_\gamma H_\gamma - \rho_\alpha H_\alpha)T^* + \mathcal{J}'(T) = 0 & \boldsymbol{x}\in\Omega,\ t\in\mathcal{I}\\ T^* = 0 & \boldsymbol{x}\in\Omega,\ t = t_f\\ -\lambda\nabla T^*\cdot\boldsymbol{n} = hT^* & \boldsymbol{x}\in\partial\Omega_1,\ t\in\mathcal{I}\\ \nabla T^*\cdot\boldsymbol{n} = 0 & \boldsymbol{x}\in\partial\Omega_2\cup\Omega_3,\ t\in\mathcal{I}\\ -\lambda\nabla T^*\cdot\boldsymbol{n} = 4\epsilon\sigma T^3T^* & \boldsymbol{x}\in\partial\Omega_4,\ t\in\mathcal{I}\end{cases} \tag{140}$$

and the cost gradient is written as:

$$\nabla j = -\left(T - T_\infty, T^*\right)_{L_2(0,T;L_2(\partial\Omega_1))}. \tag{141}$$

10.2. Rosenbrock function – GSL BFGS C file from [15]

```
#include <stdio.h>
#include <gsl/gsl_multimin.h>

/*Rosenbrock function f(x,y)=(x-p[0])^2+p[1](x^2-y^2)^2 */
double my_f(const gsl_vector *v,void *params){
double x, y;
double *p = (double *)params;
x = gsl_vector_get(v, 0);
y = gsl_vector_get(v, 1);
return (x-p[0])*(x-p[0]) + p[1]*(x*x-y)*(x*x-y);
}

/* The gradient of f, df = (df/dx, df/dy). */
void my_df(const gsl_vector *v,void *params,gsl_vector *df){
double x, y;
double *p = (double *)params;
x = gsl_vector_get(v, 0);
y = gsl_vector_get(v, 1);
gsl_vector_set(df, 0, 2*(x-p[0])+4*p[1]*x*(x*x-y));
gsl_vector_set(df, 1,-2*p[1]*(x*x-y) );
}

/* Compute both f and df together. */
void my_fdf(const gsl_vector *x,void *params,double *f,gsl_vector *df){
*f = my_f(x, params);
my_df(x, params, df);
}

int main(void){
size_t iter = 0;
int status;
const gsl_multimin_fdfminimizer_type *T;
gsl_multimin_fdfminimizer *s;
double par[2] = { 1.0, 10.0 };
gsl_vector *x;
gsl_multimin_function_fdf my_func;
my_func.n = 2;
my_func.f = &my_f;
my_func.df = &my_df;
my_func.fdf = &my_fdf;
my_func.params = &par;
/* Starting point */
x = gsl_vector_alloc (2);
gsl_vector_set (x, 0, -1.0);
gsl_vector_set (x, 1, 1.0);
T = gsl_multimin_fdfminimizer_vector_bfgs2;
s = gsl_multimin_fdfminimizer_alloc (T, 2);
gsl_multimin_fdfminimizer_set (s, &my_func, x, 1.e-6, 1e-4);

do {
iter++;
status = gsl_multimin_fdfminimizer_iterate (s);
if (status) break;
status = gsl_multimin_test_gradient (s->gradient, 1e-3);
if (status == GSL_SUCCESS) printf ("Minimum found at:\n");
printf ("%5d %.5f %.5f %10.5f\n", iter,
gsl_vector_get (s->x, 0),
gsl_vector_get (s->x, 1),
s->f);
}
while (status == GSL_CONTINUE && iter < 100);
gsl_multimin_fdfminimizer_free (s);
gsl_vector_free (x);
return 0;
}
```