GEOPHYSICS, VOL. 81, NO. 2 (MARCH-APRIL 2016); P. F11–F25, 9 FIGS., 3 TABLES.
10.1190/GEO2015-0031.1

The SEISCOPE optimization toolbox: A large-scale nonlinear optimization library based on reverse communication

Ludovic Métivier[1] and Romain Brossier[2]

ABSTRACT

The SEISCOPE optimization toolbox is a set of FORTRAN 90 routines, which implement first-order methods (steepest-descent and nonlinear conjugate gradient) and second-order methods (*l*-BFGS and truncated Newton), for the solution of large-scale nonlinear optimization problems. An efficient line-search strategy ensures the robustness of these implementations. The routines are proposed as black boxes easy to interface with any computational code, where such large-scale minimization problems have to be solved. Traveltime tomography, least-squares migration, or full-waveform inversion are examples of such problems in the context of geophysics. Integrating the toolbox for solving this class of problems presents two advantages. First, it helps to separate the routines depending on the physics of the problem from the ones related to the minimization itself, thanks to the reverse communication protocol. This enhances flexibility in code development and maintenance. Second, it allows us to switch easily between different optimization algorithms. In particular, it reduces the complexity related to the implementation of second-order methods. Because the latter benefit from faster convergence rates compared to first-order methods, significant improvements in terms of computational efforts can be expected.

INTRODUCTION

Nonlinear optimization problems are ubiquitous in geophysics. Methods such as traveltime tomography (Nolet, 1987), least-squares migration (Lambaré et al., 1992; Nemeth et al., 1999), or full-waveform inversion (FWI) (Virieux and Operto, 2009) are based on the minimization of an objective function, which measures the misfit between simulated and observed data. The simulated data (traveltime, partial measurement of the pressure, and/or displacement velocity wavefields) depend on subsurface parameters (P- or S-wave velocities, density, anisotropy parameters, and reflectivity, among others). Minimization of the objective function consists in computing the subsurface parameters, which best explain the observed data. In the general case, the simulated data depend nonlinearly on the subsurface parameters, yielding a nonlinear optimization problem.

Two general classes of methods exist to solve this type of problem: global and semiglobal (Sen and Stoffa, 1995; Spall, 2003) or local descent algorithms (Bonnans et al., 2006; Nocedal and Wright, 2006). The methods belonging to the first class have the capability to find the global minimum of an objective function from any given starting point. They are said to be globally convergent. These methods proceed in a guided-random sampling of the parameter space to find the global minimum of the objective function. However, as soon as the number of discrete parameters exceeds a few tenths to hundreds, the number of evaluations of the objective function required to find its global minimum becomes excessively important for these methods to be applied in a reasonable time, even using modern high-performance computing (HPC) platforms. In geophysics, realistic applications yield optimization problems for which the number of unknown parameters is easily several orders of magnitude higher (billions may be involved for 3D FWI applications), rendering global approaches intractable.

Local descent algorithms only guarantee local convergence: From a given starting point, the nearest local minimum of the objective function is found. Instead of proceeding to a random sampling, these methods are based on a guided exploration of the model space following the descent directions of the objective function. From an initial guess, an estimation of the local minimum is found by successive updates following these directions. The convergence and the monotonic decrease of the objective function is guaranteed by a proper line-search or trust-region strategy (Nocedal and Wright, 2006). This

Peer-reviewed code related to this article can be found at http://software.seg.org/2016/0001.
Manuscript received by the Editor 16 January 2015; revised manuscript received 23 September 2015; published online 18 February 2016.
[1]LJK, University of Grenoble Alpes, CNRS and ISTerre, Grenoble, France. E-mail: ludovic.metivier@ujf-grenoble.fr.
[2]ISTerre, University of Grenoble Alpes, Grenoble, France. E-mail: romain.brossier@ujf-grenoble.fr.

consists in giving an appropriate scale to the update brought to the current model estimate. This guided exploration strategy is drastically cheaper than a random sampling of the model space: The number of iterations required to converge to a minimum is far lower than the number of evaluations of the objective function, which would be required to obtain an acceptable sampling of the model space when its dimension is large. This is the reason why local descent methods are used for solving large-scale optimization problems.

Among possible local descent methods, the simplest ones use only the first-order derivatives of the objective function (the gradient) to define the descent direction. These methods are referred to as first-order methods. The steepest-descent and nonlinear conjugate gradient algorithms are examples of these methods. More elaborate descent schemes account for the curvature of the objective function through quasi-Newton algorithms. These algorithms are based on approximations of the inverse Hessian operator (the matrix of the second-order derivatives of the objective function) following three different (possibly complementary) strategies.

The first strategy is referred to as the preconditioning strategy. In some situations, an approximation of the inverse Hessian operator can be computed from analytic or semianalytic formulas. Diagonal approximations can be used. An example in the context of FWI is the pseudo-Hessian preconditioning strategy proposed by Shin et al. (2001). This strategy can be used to derive an approximate inverse of the Hessian operator, which is integrated within first-order algorithms by multiplying the gradient by this approximation. The second strategy is based on the computation of an approximate inverse Hessian operator, from finite differences of the gradient, such as in the l-BFGS algorithm (Nocedal, 1980). Instead of using an approximate inverse of the Hessian operator, the third strategy, which is referred to as the truncated Newton method, incompletely solves the linear system associated with the Newton equations, namely, the linear system relating the gradient to the descent direction, through a conjugate gradient iterative solver (Nash, 2000).

In geophysics, the solution of minimization problems is often computed using basic methods such as the steepest-descent or the nonlinear conjugate gradient. In addition, line-search algorithms are not always implemented, and as a consequence, the convergence and monotonic decrease of the objective function are not guaranteed. Moreover, it is not unusual to see computer codes in which the minimization procedure is embedded into the routines related to the physical problem at stake. This type of implementation is not flexible because it implies that the optimization process has to be redeveloped for each application. Improving the optimization process from first-order to second-order methods may also require significant modifications of the code.

The SEISCOPE optimization toolbox has been designed to propose a solution to these issues. The first objective of the toolbox is to propose minimization routines that can be easily interfaced with any computational code through a reverse communication protocol (Dongarra et al., 1995). The principle is the following: The computation of the solution of the optimization problem is performed in a specific routine of the code. This routine is organized as a minimization loop. At each iteration of the loop, the minimization routine from the toolbox chosen by the user is called. This routine communicates what is the quantity required at the current iteration: objective function, gradient, or Hessian-vector product. These quantities are computed by the user in specific routines, external to the minimization loop. The process continues until the convergence is reached. This implementation paradigm yields a complete separation between the code related to the physics of the problem and the code related to the solution of the minimization problem. This ensures a greater versatility because one can easily modify one of these parts while keeping the other unchanged.

The second objective of the toolbox is to propose robust and efficient methods for large-scale nonlinear optimization problems. Four different optimization schemes are implemented: the steepest-descent, nonlinear conjugate gradient, l-BFGS, and truncated Newton methods. These methods share the same line-search strategy, which assures the convergence and the monotonic decrease of the objective function and allows for comparison of the efficiency of the methods on a given application. In addition, the four methods can be combined with preconditioning strategies: Any information regarding the curvature of the objective function can be easily incorporated to improve the convergence rate of the algorithms. Finally, the toolbox offers the possibility to use second-order methods as well as first-order methods. The complexity associated with the implementation of the l-BFGS approximation is hidden to the user. The use of truncated Newton methods only requires us to implement Hessian-vector products.

It should be mentioned that similar developments in the `C++` language have been provided by members of the Rice project. The developments are based on the Rice vector library. The available methods are l-BFGS and the truncated Newton method implemented in the framework of the trust-region globalization strategy, also known as the Steihaug-Toint algorithm (Steihaug, 1983; Gould et al., 1999). However, to the best of our knowledge, the implementation is not based on a reverse communication protocol.

We devote this article to the presentation of the SEISCOPE optimization toolbox. First, we give an overview of the optimization methods that are implemented. No details on convergence proofs and computation of convergence rates are given. Those can be found in reference textbooks (Bonnans et al., 2006; Nocedal and Wright, 2006). We focus on the general principles of these methods. We present a code sample in which the toolbox is interfaced to illustrate the reverse communication protocol in a practical example, and we provide details on the implementation of the routines. In the "Numerical study" section, we investigate the use of the toolbox on two different applications. We minimize the bidimensional `Rosenbrock` function using the different methods proposed in the toolbox. We compare the convergence of these algorithms and the optimization path they follow. We give a second illustration on a large-scale nonlinear minimization problem related to a 2D acoustic frequency-domain FWI case study. We consider synthetic data acquired on the Marmousi 2 model. We compare the convergence of the different minimization algorithms. These two examples emphasize the superiority of second-order optimization methods over first-order methods. We give a conclusion and perspectives in the final section.

THEORY

Generalities

The routines implemented in the SEISCOPE optimization toolbox are designed to solve unconstrained and bound constrained nonlinear minimization problems under the general form

$$\min_{x\in\Omega} f(x), \tag{1}$$

where

$$\Omega = \prod_{i=1}^{n} [a_i, b_i] \subset \mathbb{R}^n, \quad n \in \mathbb{N}, \tag{2}$$

and $f(x)$ is a sufficiently smooth function (at least twice differentiable) depending nonlinearly on the variable x.

All the routines considered in the toolbox belong to the class of local descent algorithms. From an initial guess $x_0 \in \Omega$, a sequence of iterates is built following the recurrence

$$x_{k+1} = x_k + \alpha_k \Delta x_k, \tag{3}$$

where $\alpha_k \in \mathbb{R}_+^*$ is a step length and $\Delta x_k \in \mathbb{R}^n$ is a descent direction. The recurrence (equation 3) is repeated until a certain stopping convergence criterion is reached.

The step length α_k is computed through a line-search process to satisfy the standard Wolfe conditions. ("Standard" is understood as being opposed to the strong Wolfe conditions, which are more restrictive. This issue is important regarding the nonlinear conjugate gradient method implementation [see the "Nonlinear conjugate gradient" subsection]. The Wolfe conditions are described later in equations 13 and 14.) Satisfying the Wolfe conditions ensures convergence toward a local minimum, provided $f(x)$ is bounded and sufficiently smooth (Nocedal and Wright, 2006). The satisfaction of the bound constraints is ensured through the projection of each iterate within the feasible domain Ω in the line-search process.

The different nonlinear minimization methods differ by the computation of the descent direction Δx_k. First, a description of the computation of these quantities depending on the minimization routines is given. More details on the line-search and the bound constraints enforcement algorithm are given after.

In the following, the gradient of the objective function is denoted by $\nabla f(x)$, and the Hessian operator by $H(x)$. A preconditioner for the Hessian operator $H(x)$ is denoted by $P(x)$, such that

$$P(x) \simeq H^{-1}(x). \tag{4}$$

Steepest descent

The simplest optimization routine implemented is the steepest-descent algorithm. This method uses the following descent direction:

$$\Delta x_k = -P(x_k)\nabla f(x_k). \tag{5}$$

The convergence rate of this method is linear. When no preconditioner is available (P is the identity matrix), the descent direction is simply the opposite of the gradient. In this case, examples show that the number of iterations required to reach the convergence can be extremely high.

Nonlinear conjugate gradient

The conjugate gradient algorithm is an iterative linear solver for symmetric positive definite systems. This method can be interpreted as a minimization algorithm for quadratic functions. The nonlinear conjugate gradient method is conceived as an extension of this algorithm to the minimization of general nonlinear functions. The model update is computed as a linear combination of the opposite of the gradient and the descent direction computed at the previous iteration

$$\begin{cases} \Delta x_0 = -P(x_0)\nabla f(x_0), \\ \Delta x_k = -P(x_k)\nabla f(x_k) + \beta_k \Delta x_{k-1}, \quad k \geq 1, \end{cases} \tag{6}$$

where $\beta_k \in \mathbb{R}$ is a scalar parameter. Different formulations can be used to compute β_k, giving as many versions of the nonlinear conjugate gradient algorithm. Standard implementations use formulas from Fletcher and Reeves (1964), Hestenes and Stiefel (1952), or Polak and Ribière (1969). In the SEISCOPE optimization toolbox, an alternative formula, proposed by Dai and Yuan (1999), is used. In this particular case, the scalar β_k is computed as

$$\beta_k = \frac{\|\nabla f(x_k)\|^2}{(\nabla f(x_k) - \nabla f(x_{k-1}))^T \Delta x_{k-1}}. \tag{7}$$

This formulation ensures the convergence toward the nearest local minimum as soon as the steplength satisfies the Wolfe conditions. This is not the case for the standard formulations, which require satisfying the strong Wolfe conditions. This is the reason why the formulation from Dai and Yuan (1999) is preferred. This allows us to use the same line-search strategy for all the optimization routines proposed in the toolbox, including the nonlinear conjugate gradient. The convergence of the nonlinear conjugate gradient is linear, as well as for the steepest-descent algorithm. In some cases, the reduction of the number of iterations can be significant; however, this potential acceleration is case dependent, and therefore unpredictable.

Quasi-Newton *l*-BFGS method

Among the class of quasi-Newton methods, the l-BFGS algorithm has become increasingly popular because of its simplicity and efficiency (Nocedal, 1980). The l-BFGS approximation relies on an approximate inverse Hessian operator Q_k computed through finite differences of l previous values of the gradient. The resulting descent direction is computed as

$$\Delta x_k = -Q_k \nabla f(x_k). \tag{8}$$

Equation 8 is noteworthy in that it is similar in form to equation 5, where Q_k plays the role of the preconditioner $P(x_k)$. The difference comes from the fact that the preconditioning matrix Q_k is computed in a systematical way, only from the objective function gradient values, and is updated at each iteration following the l-BFGS formula. In the preconditioned steepest-descent algorithm, the user is in charge to provide the preconditioning matrix $P(x_k)$. The l-BFGS method achieves a superlinear convergence rate, and it is usually faster than the nonlinear conjugate gradient and steepest-descent algorithms.

In terms of practical implementation, matrix Q_k is never explicitly built. The product $Q_k \nabla f(x_k)$ is directly computed instead, following a two-loop recursion algorithm (Nocedal and Wright, 2006). Interestingly, it is possible to incorporate an estimation of the inverse Hessian operator through the preconditioner $P(x_k)$ into this two-loop recursion. This means that the information from the

preconditioner $P(x_k)$ and the l-BFGS formula can be combined to approximate, as accurately as possible, the action of the inverse Hessian operator. Surprisingly, this possibility is rarely mentioned in the literature, whereas it seems to be one of the most efficient strategy for large-scale nonlinear optimization problems. In the context of FWI, the combination of a diagonal preconditioner and the l-BFGS approximation can be shown to yield the best computational efficiency among the different optimization strategies available from the toolbox (see the numerical results on the Marmousi 2 case study). The details of this approach are given in Appendix A.

Truncated Newton method

Instead of using an approximate inverse of the Hessian operator, the truncated Newton method computes an inexact (truncated) solution of the linear system associated with the Newton equation (Nash, 2000):

$$H(x_k)\Delta x_k = -\nabla f(x_k). \tag{9}$$

This approximate solution of the linear system is computed through a matrix-free conjugate gradient algorithm: Only the action of the Hessian operator on a given vector is required. In the context of FWI, the gradient is usually computed through first-order adjoint methods. Second-order adjoint strategies can be used to compute the action of the Hessian on a given vector (Métivier et al., 2012). In both cases, the computational complexity is reduced to the solution of the wave equation for different source terms. This allows us to use truncated Newton methods for solving FWI problems at a reasonable computational cost. This strategy has been investigated in Métivier et al. (2013, 2014) and Castellanos et al. (2015).

The stopping criterion for the inner linear system is

$$\|H(x_k)\Delta x_k + \nabla f(x_k)\| \leq \eta_k \|\nabla f(x_k)\|, \tag{10}$$

where η_k is a forcing term. The equation 10 reveals that η_k controls the accuracy of the relative residuals of the linear system (equation 9). For large-scale nonlinear applications, particular choices for η_k are recommended, as is detailed by Eisenstat and Walker (1996). Because the targeted applications for this toolbox are precisely large-scale nonlinear minimization problems, the default implementation of the computation of η_k is based on this work: η_k should decrease as soon as the function that is minimized is locally quadratic. The measure of the local quadratic approximation of the misfit function is based on the gradient current and previous values. For a FWI application, this strategy reveals to be efficient (Métivier et al., 2013, 2014; Castellanos et al., 2015). However, for simpler problems, a constant and small value for η_k might produce better results. In numerical results section, for the `Rosenbrock` experiment, η_k is set to 10^{-5} and remains constant throughout the iterations.

In the context of the truncated Newton method, the preconditioner $P(x)$ is used naturally as a preconditioner for the linear system (equation 9). This implies that the symmetry of equation 9 has to be preserved through the preconditioning operation. Therefore, $P(x)$ cannot be used simply as a left or right preconditioner. The preconditioner $P(x)$ has to be symmetric definite positive. In this case, it can be factorized as

$$P(x) = C(x)^T C(x), \tag{11}$$

and the following symmetric preconditioning strategy can be used:

$$\begin{aligned} C(x_k)^T H(x_k) C(x_k) \Delta y_k &= -C(x_k)^T \nabla f(x_k), \\ \Delta x_k &= C(x_k)\Delta y_k. \end{aligned} \tag{12}$$

The symmetry of equation 9 is preserved because $C(x_k)^T H(x_k) C(x_k)$ is symmetric by construction.

In terms of implementation, the formulation 12 may imply that a factorization of the preconditioner $P(x)$ under the form (equation 11) is required. However, a particular implementation of the preconditioned conjugate gradient algorithm, such as the one proposed in Nocedal and Wright (2006), allows us to solve equation 12 through the conjugate gradient method using only matrix-vector products from $H(x)$ and $P(x)$. In practice, this preconditioned conjugate gradient implementation differs from the nonpreconditioned conjugate gradient implementation only by the multiplication of the residuals by the matrix $P(x)$. Thus, from the user's point of view, only the computation of the action of $P(x)$ on a given vector is required.

The truncated Newton method has been shown to converge even faster than the l-BFGS method, achieving a nearly quadratic convergence rate close to the solution. However, the computation cost of each iteration is increased by the (inexact) solution of the Newton equation. In practice, the l-BFGS and truncated Newton methods have been shown to be competitive on a collection of benchmark problems from the nonlinear optimization community (Nash and Nocedal, 1991). The same conclusions have been drawn for monoparameter FWI applications (Métivier et al., 2013, 2014; Castellanos et al., 2015), and this is what can be observed in the numerical experiments proposed in this study.

Line search and bound constraints

Using a line-search strategy is necessary to ensure the robustness of the optimization routines, that is, guaranteeing a monotonic decrease of the objective function and the convergence toward a local minimum. The line-search algorithm enforces the Wolfe conditions. These are the sufficient decrease conditions

$$f(x_k + \alpha\Delta x_k) \leq f(x_k) + c_1 \alpha \nabla f(x_k)^T \Delta x_k, \tag{13}$$

and the curvature condition

$$\nabla f(x_k + \alpha\Delta x_k)^T \Delta x_k \geq c_2 \nabla f(x_k)^T \Delta x_k, \tag{14}$$

where c_1 and c_2 are coefficients, such that

$$0 < c_1 < c_2 \leq 1. \tag{15}$$

In practice, the values

$$c_1 = 10^{-4}, \qquad c_2 = 0.9, \tag{16}$$

are used, following Nocedal and Wright (2006). The first condition indicates that the steplength α should be computed to ensure that the update in the direction Δx_k will generate a sufficient decrease of the objective function. The second condition is used to rule out too-small values for α itself, which would lead to undesirable small updates of the estimation x_k.

The bound constraints satisfaction is enforced through a projection method, similarly to the strategy proposed by Byrd et al. (1995). At each iteration of the line search, before testing the two Wolfe conditions, the updated model $x_k^* = x_k + \alpha \Delta x_k$ is projected into the feasible domain Ω through the operator $\mathrm{Proj}(x)$ defined as

$$\mathrm{Proj}(x)_i = \begin{cases} x_i & \text{if } a_i \leq x_i \leq b_i, \\ a_i & \text{if } x_i < a_i, \\ b_i & \text{if } x_i > b_i, \end{cases} \tag{17}$$

where the subscript i denotes the ith component of the vectors of $\mathbb{R}^n$. This procedure ensures that the estimated model always remains in the feasible domain Ω. Although there is no proof of convergence for optimization algorithms using this bound constraints enforcement, the method has yielded satisfactory results in practice. In some sense, it can be viewed as a simplification of the method of projection on convex sets proposed in the context of seismic inversion by Baumstein (2013).

IMPLEMENTATION

The reverse communication protocol, on which all the routines of the SEISCOPE optimization toolbox are based, requires a specific implementation of the different algorithms. The principle of reverse communication from the user's point of view is first introduced. A specific code example is used, in which the minimization of a function is performed with the preconditioned l-BFGS routine from the toolbox. Next, details on the implementation of the algorithms of the toolbox are given. The general algorithmic structure of the different optimization methods in the toolbox is presented, as well as the structure of the line-search algorithm. Note that in the current version, all the toolbox routines are implemented in single precision, with the purpose to save computational time on large-scale applications.

Reverse communication protocol

The principle of reverse communication is illustrated from the code sample presented in Figure 1. In this example, the user minimizes an objective function $f(x)$, which corresponds to the `Rosenbrock` function. In this example, x is a 2D real vector, such that $x \in \mathbb{R}^2$. In general, x is an n-dimensional vector, which is implemented in an array structure. The minimization is performed from an initial value x_0, which initializes x. At the end of the minimization loop, x contains the solution of the minimization problem. In this example, the minimization is performed with the preconditioned version of the l-BFGS algorithm proposed in the toolbox by the routine `PLBFGS`.

The code is organized in a do `while` loop. The principle is as follows: While the convergence has not been reached, further iterations of the minimization process are needed. Each iteration of this loop starts by a call to the optimization routine. The communication with the optimization routine is achieved through the four-character chain `FLAG`. This character chain is an input/output variable of the optimization routine. On return of each call, it is updated to communicate with the user and to require the computation of particular quantities. On first call, this variable is initialized by the user to the chain of characters "`INIT`."

Following this principle, after each call to the optimization routine, the user tests the possible values for the communicator and performs the subsequent actions. If `FLAG` has been set to "`GRAD`," the computation of the objective function and its gradient at the current value of `x` is required. In this case, the code calls the subroutine `Rosenbrock` with the current `x` as an input. On return, the subroutine `Rosenbrock` computes the objective function as well as its gradient at the current `x`, and it stores these quantities in the variables `fcost` and `grad`, respectively. The code reaches the end of the `if/end if` loop and returns to the minimization function `PLBFGS` with the required values for the variables `fcost` and `grad`.

If `FLAG` is set to "`PREC`," the optimization loop requires the preconditioner to be applied. In this case, the user extracts the vector `y` from the variable `optim`. The data type of `optim` is proper to the optimization toolbox and is described in the file `optim.h`. The field of interest is `optim%q_plb`. The user then applies a preconditioner to `y` through the call to the routine `apply_Preco`. On return of this routine, the quantity `Py` is transferred into the data structure `optim` in the field `optim%q_plb`. The code reaches

```
do while ((FLAG.ne.'CONV').and.(FLAG.ne.'FAIL'))

   call PLBFGS(n,x,fcost,grad,grad_preco,optim,FLAG)

   if(FLAG.eq.'GRAD') then

      !compute objective function and its gradient at point x

      call Rosenbrock(x,fcost, grad)

   endif

   if(FLAG.eq.'PREC') then

      !apply preconditioning to optim%q_plb

      y(:)=optim%q_plb(:)

      call apply_preco(y,Py)

      optim%q_plb(:)=Py(:)

   endif

   if(FLAG.eq.'NSTE') then

      !a new step has been taken

      call print_information(x)

   endif

enddo
```

Figure 1. Example of a reverse communication loop.

the end of the `if/end if` loop and returns to the minimization function `PLBFGS` with the required values for the variable `optim%q_plb`. A more efficient strategy in terms of HPC would consist in performing the two copies involving the vector `optim%q_plb` directly in the routine `apply_Preco` to preserve the locality of the data. However, the purpose here is to illustrate the use of the toolbox rather than proposing an optimized implementation.

When `FLAG` is set to "`NSTE`," this is an indication that the line-search process just terminated and has yielded a new iterate x_{k+1} of the minimization sequence. This new iterate is stored in the variable `x`. This information can be useful to track the history of the computed approximation of the solution. In this example, the user calls a routine `print_information` for this purpose.

If `FLAG` is "`CONV`," the default convergence criterion implemented in the optimization routine has been reached. This convergence criterion is based on the reduction of the relative objective function. The convergence is declared when the condition

$$\frac{f(x_k)}{f(x_0)} \leq \varepsilon, \tag{18}$$

is satisfied, where ε is a threshold parameter initialized by the user.

If `FLAG` is "`FAIL`," a failure in the line-search process has been detected. The user is in charge of defining the minimization loop and is responsible for defining the actual stopping criterion. This stopping criterion can be based on the information provided by the optimization routine. For instance, in the example provided, the minimization loop stops as soon as `FLAG` is set to "`CONV`" or to "`FAIL`." However, it is also possible to ignore this information or to complete it with additional tests to define the actual stopping criterion. On exit of the `do while` loop, `x` contains the solution reached when the convergence criterion is satisfied.

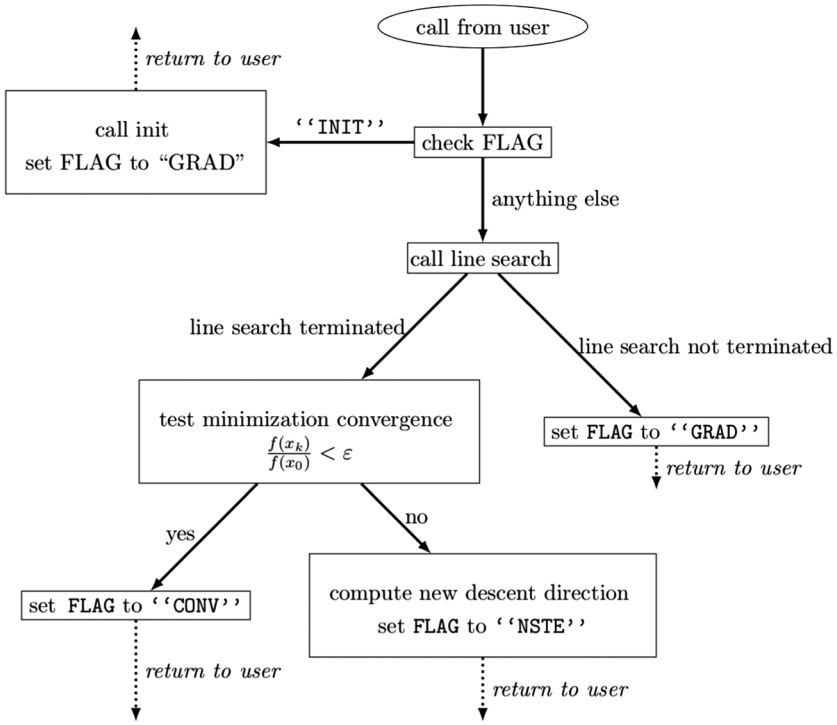

Figure 2. General structure of the toolbox optimization routines.

The code sample presented in this example can be used as a template for the minimization of any objective function $f(x)$ for which it is possible to compute the gradient. The optimization algorithm used here is the preconditioned l-BFGS method; however, this code sample can be straightforwardly extended to the use of the other methods from the toolbox. To this purpose, Table 1 presents the different values that can be taken by the variable `FLAG` and the corresponding actions required depending on the choice of the minimization routine. For practical use, more details are given in the toolbox documentation.

In the following subsections, details on the implementation of the toolbox are given. Readers only interested in the use of the toolbox may skip the two following subsections and jump directly to the "Numerical examples" section.

General algorithmic structure of the optimization routines

The diagram in Figure 2 presents the general structure of the toolbox optimization routines. The workflow starts with a call of the routine from the user. The value of the communicator `FLAG` is first tested.

If it is equal to "`INIT`," this means that this is a first call from the user. In this case, subsequent initializations have to be performed. An initialization routine (here `init`) is thus called. This routine is not described in detail here; this is where memory allocation and parameters settings (such as the scalars c_1 and c_2 for the line search) are performed. The memory allocation consists in allocating different fields in the data structure `optim`. Depending on the minimization routines, certain fields may be allocated or not. The l-BFGS algorithm requires specific fields related to the l-BFGS approximation to be allocated for instance. After this initialization has been performed, the communicator `FLAG` is set to "`GRAD`" and the routine returns to the user.

If the communicator `FLAG` has any value different from "`INIT`," the line-search algorithm is called. Returning from the line-search, the algorithm asks if the line search has terminated or not.

If the line search has terminated, this means that a new iterate has been computed. This new iterate is stored in the variable `x`. In this situation, the algorithm tests if the stopping criterion (equation 18) is satisfied. If this is the case, the communicator `FLAG` is set to "`CONV`" and the routine returns to the user. If not, this means that a new step has been taken in the minimization process, whereas the convergence has not been reached yet. In this case, a new descent direction is computed, and the communicator `FLAG` is set to "`NSTE`." This value of the communicator is useful for intermediate printing of the solution for instance. The routine returns to the user.

If the line-search process is not terminated, this means that a new value of the objective function and its gradient are required to continue the line-search process. In this case, the step length is updated, the communicator `FLAG` is set to "`GRAD`," and the routine returns to the user.

The steepest-descent, nonlinear conjugate gradient, and l-BFGS algorithms share exactly this

algorithmic structure in the toolbox implementation. The only difference is related to the routine called for the initialization and the computation of the descent direction.

The structure has to be modified as soon as the computation of the descent direction also requires a direct intervention of the user. This is the case for instance for the preconditioned l-BFGS routine. In this situation, the computation of the descent direction is split into two parts, corresponding to the two-loop recursion (see Appendix A for more details). Between these two loops, a call to the user is required for applying the preconditioner.

The same is true for truncated Newton methods (with or without preconditioning). The computation of the descent direction requires additional communications with the user for applying the Hessian operator and the preconditioner to particular vectors, to compute the incomplete solution of the system (equation 12).

From this overview of the algorithmic structure, one can see that the line-search algorithm plays a central role in the optimization routine's implementation. More details on its implementation are given in the next subsection.

Line-search implementation

A diagram summarizing the line-search algorithmic structure is presented in Figure 3. The structure is also based on a reverse communication paradigm. The calling program is in charge of the computation of the quantities required by the line-search algorithm: the objective function and gradient computed at particular points.

The first call to the line-search subroutine is indicated by an internal line-search flag. In this case, a first candidate for the next model update x_{k+1} is computed. This candidate is denoted by x_k^*, computed as

$$x_k^* = \text{Proj}(x_k + \alpha_k \Delta x_k). \tag{19}$$

The formula 19 means that the candidate is computed as the current value x_k updated in the descent direction Δx_k with a step length α_k and projected into the set Ω to satisfy the bound constraints, which could be potentially imposed.

At this stage, the Wolfe conditions have to be tested on x_k^*. However, this requires the computation of the quantities $f(x_k^*)$ and $\nabla f(x_k^*)$. The line-search routine thus goes back to the calling program with a request for computing these quantities. In particular, it indicates that the line-search did not terminate.

When these quantities have been computed, the calling program goes back to the line-search routine. This time, the internal line-search flag indicates it is not a first call to the line search. As a consequence, the two Wolfe conditions are tested in a sequential order. If they are satisfied, the line-search algorithm terminates successfully with a new model update x_{k+1}, which is set to the candidate value x_k^*. The internal line-search flag is set back to "`first_call`."

If one of the two Wolfe conditions is not satisfied, the steplength α_k has to be adjusted. The modification of α_k is performed following the bracketing strategy proposed in Bonnans et al. (2006). Once α_k has been modified, a new candidate x_k^* is computed, and the line-search algorithm goes back to the calling program with a request for the computation of $f(x_k^*)$ and $\nabla f(x_k^*)$.

The optimization routines implemented in the toolbox are thus based on a two-level reverse communication process. The first level is between the user and the optimization routine. The second level is embedded in the optimization routine itself and corresponds to the line-search implementation. The potential complexity of such an implementation is transparent for the user. When asked to compute the gradient of the objective function, the user does not have to know if this computation is requested for the line-search process or the evaluation of a new descent direction.

In terms of practical implementation, the line-search algorithm requires us to define an initial step length α_0. In the current implementation, this step length is set to one. Besides, at each iteration k of the minimization, the new step length α_k is initialized to the previous value α_{k-1}. From our experience, this approach results in the following behavior: At the first iteration of the minimization process, the line-search algorithm may require several internal iterations to adjust the step length. However, for further iterations, the step length computed at the previous iteration often directly yields a candidate that satisfies the Wolfe conditions. The additional computational cost related to the line search is thus limited to the first iteration of the minimization. Note that this cost can be reduced by the user by using an appropriate scaling of the objective function. Indeed, at the first iteration, for most optimization methods, the descent direction is equal to the gradient. The step-length adjustment from the line search thus corresponds to an automatic scaling of the misfit function. This process can be accelerated by a proper scaling performed directly by the user.

Summary of the routines implemented in the toolbox

Table 2 summarizes the name of the minimization routines implemented in the toolbox together with their basic properties. Although four main minimization methods are presented, the toolbox proposes a total of six subroutines. This is due to the implementation of preconditioning. Although it is straightforward for the steepest-descent and nonlinear conjugate gradient methods, the implementation of preconditioning for the l-BFGS and truncated Newton methods requires substantial modifications. Our choice is thus to im-

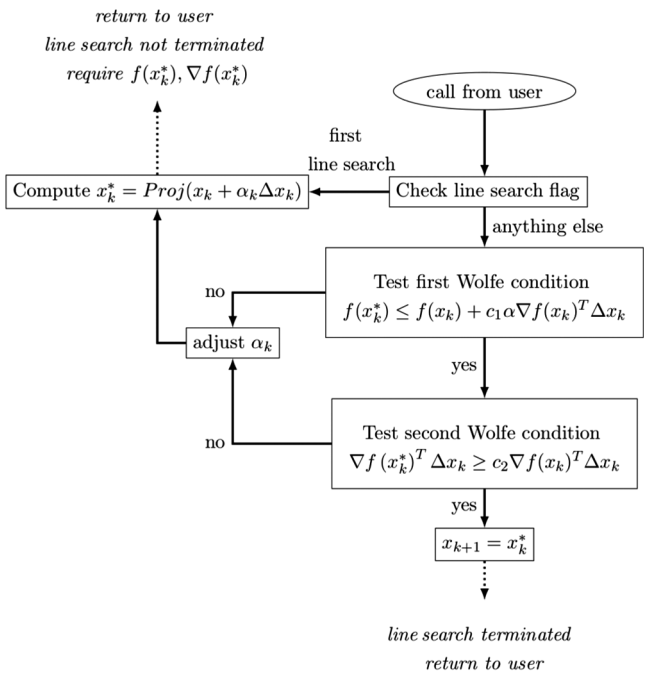

Figure 3. Schematic view of the line-search implementation.

plement a single version for the steepest-descent and nonlinear conjugate gradient methods, where the user only has to apply the identity operator if no preconditioner is available, whereas two distinct versions (with and without preconditioning) are implemented for the l-BFGS and truncated Newton methods.

In theory, first-order methods have a linear convergence rate, whereas second-order methods benefits from superlinear convergence. Although the Newton method converges with a quadratic rate close from the solution, the l-BFGS and the truncated Newton methods only benefit from a superlinear convergence rate because only an approximation of the inverse Hessian operator is used. The six routines require the computation of the gradient. Only the truncated Newton method (with and without preconditioning) requires the computation of Hessian-vector products.

Table 2 also summarizes the comparisons in terms of convergence speed (reduction of the misfit function depending on the number of iteration) and computational efficiency (reduction of the misfit function depending on the number of gradient/Hessian-vector product evaluations) observed on the two numerical examples presented in the following section.

NUMERICAL EXAMPLES

In this section, numerical examples of use of the SEISCOPE optimization toolbox are provided. First, the bidimensional `Rosenbrock` objective function is considered. This function serves as a benchmark in reference textbooks to calibrate optimization methods. Its particularity is to present a narrow valley of attraction, where the function has a very flat minimum. It is, therefore, a challenging problem for nonlinear optimization. The goal of this example is to illustrate the properties of the different optimization methods proposed in the toolbox. This example can be reproduced using the test case implemented in the toolbox (see file `00README_for_GEOPHYSICS_submission`). The second example shows an

Table 1. Summary of the different values taken by the communication variable `FLAG` and their meaning.

FLAG values	Action required/meaning
"INIT"	This flag is set only by the user, prior to the minimization loop.
	On reception of this flag, the solver performs the necessary initializations (parameter settings and memory allocations).
"CONV"	The convergence criterion $\frac{f(x_k)}{f(x_0)} < \varepsilon$ is satisfied.
	The variable `x` contains the solution x_k computed at this stage.
"FAIL"	The line search has terminated on a failure.
"NSTE"	The line search has terminated successfully.
	The variable `x` contains the new iterate x_{k+1}.
"GRAD"	Compute $f(x)$ and store it in `fcost`. Compute $\nabla f(x)$ and store it in `grad`.
	For `PNLCG` and `PSTD`, apply the preconditioner to $\nabla f(x)$ and store it in `grad_preco`.
"PREC"	Only for `PLBGS` and `PTRN` routines.
	If the routine called is `PLBFGS`, then apply the preconditioner to the variable `optim%q_plb` and store it in the same variable. If the routine called is `PTRN`, then apply the preconditioner to the variable `optim%residual` and store it in the variable `optim%residual_preco`.
"HESS"	Only for `TRN` and `PTRN` routines.
	Apply the Hessian operator to the variable `optim%d` and store it in `optim%Hd`.

Table 2. Summary of the minimization routines implemented in the toolbox. The ranks in the two last lines correspond to the performance on numerical examples 1 (`Rosenbrock` function) and 2 (FWI on the Marmousi mode) in terms of convergence rate (normal font), and the number of computed gradients (bold font).

	First-order methods		Second-order methods			
Routine name	PSTD	PNLCG	LBFGS	PLBFGS	TRN	PTRN
Method	Steepest descent	Nonlinear CG	l-BFGS	l-BFGS	Truncated Newton	Truncated Newton
Preconditioning	Yes	Yes	No	Yes	No	Yes
Convergence rate	Linear	Linear	Superlinear	Superlinear	Superlinear	Superlinear
$\nabla f(m)$ required	Yes	Yes	Yes	Yes	Yes	Yes
$H(m)v$ required	No	No	No	No	Yes	Yes
Rank test 1	4,**4**	3,**3**	2,**1**	N/A	1,**2**	N/A
Rank test 2	3,**3**	4,**4**	N/A	2,**1**	N/A	1,**2**

application of 2D acoustic frequency-domain FWI on the Marmousi 2 model. The Marmousi 2 model is a standard benchmark for today FWI algorithms (Martin et al., 2006). The interest of this example is to illustrate the capability of the SEISCOPE optimization toolbox to handle realistic scale optimization problems and to perform comparisons between the different algorithms that are proposed. In this case, the number of discrete unknowns reaches almost 100,000. This example is not directly proposed in the toolbox code attached to this submission. However, it could be reproduced using the open-source code from the SEISCOPE team `TOY2DAC` (Seiscope, 2015). The Marmousi 2 model can be downloaded from AGL (2015).

The `Rosenbrock` function

The analytic expression for the bidimensional `Rosenbrock` function is

$$f(x,y) = (1-x)^2 + 100(y-x^2)^2. \tag{20}$$

The gradient and the Hessian of the `Rosenbrock` function are, respectively

$$\nabla f(x,y) = \begin{pmatrix} 2(x-1) - 400x(y-x^2) \\ 200(y-x^2) \end{pmatrix}, \tag{21}$$

$$H(x,y) = \begin{pmatrix} 1200x^2 - 400y + 2 & -400x \\ -400x & 200 \end{pmatrix}. \tag{22}$$

The `Rosenbrock` function reaches its minimum value at the point $(1,1)$. In Figure 4, the valley of attraction appears in blue. The steepness of the objective function in this valley is weak, which renders the convergence to the minimum $(1,1)$ difficult for local descent algorithms.

As an illustration of the performance of the algorithms proposed in the toolbox, the different paths taken by steepest-descent, nonlinear conjugate gradient, l-BFGS, and truncated Newton methods are presented in Figure 5. The starting point (initial guess) is taken as $(x_0, y_0) = (0.25, 0.25)$.

The memory parameter l, which controls the number of gradient stored to build the l-BFGS approximation of the inverse Hessian, is set to 20. Changes of this value do not show significant modification of the l-BFGS path. For realistic-size case studies, 20 is already a "large" value for this parameter. For strongly nonlinear misfit functions (for instance, in the presence of noisy data when the method is applied to nonlinear least-squares minimization), it may be useful to take lower values such that $l = 3$ or $l = 5$, as the information carried out by older iterates may not be relevant to the local approximation of the inverse Hessian operator.

Starting from (x_0, y_0), all the methods head rapidly toward the valley of attraction. The path they take when they reach this area is, however, very different. The steepest-descent algorithm performs the worst. It generates very small steps, and therefore numerous accumulation points appear on the path it takes. The convergence is reached only after several thousand iterations. The nonlinear conjugate gradient performs

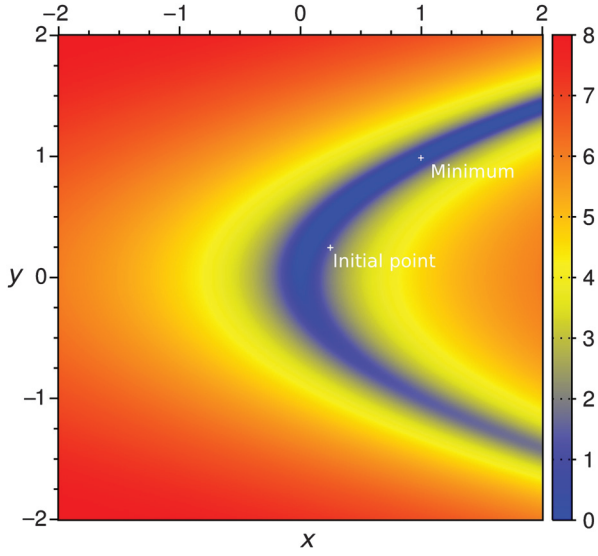

Figure 4. Bidimensional `Rosenbrock` function map. The two white crosses highlight the starting point for the optimization process located at $(0.25, 0.25)$ and the minimum of the `Rosenbrock` function, at the point $(1,1)$. The valley of attraction appears as a broad blue channel, where the function has a very flat minimum.

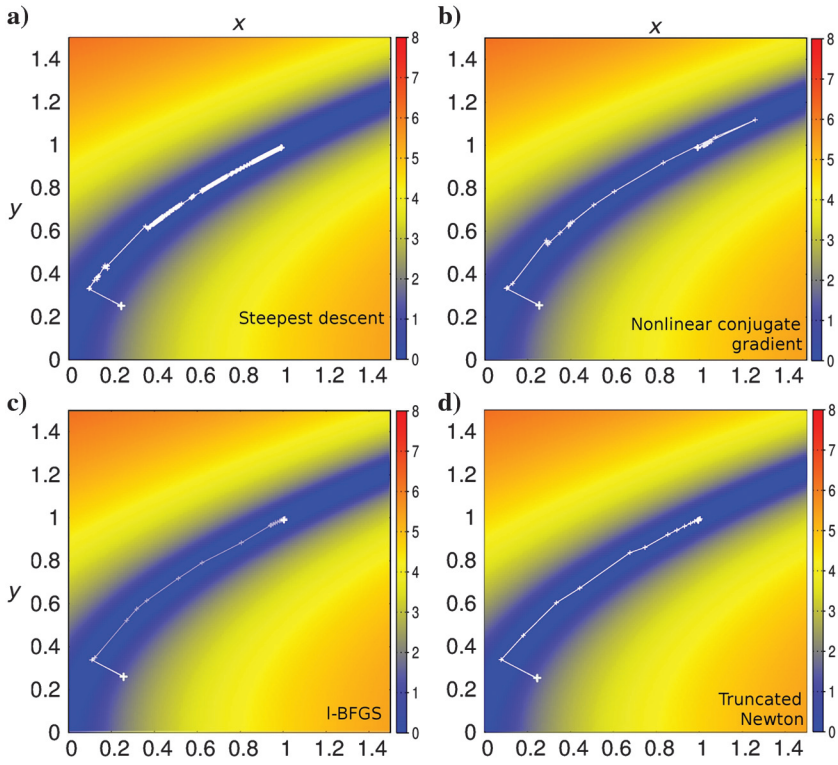

Figure 5. Convergence paths taken by the four different optimization methods: (a) steepest-descent, (b) nonlinear conjugate gradient, (c) l-BFGS, and (d) truncated Newton. The starting point is $(0.25, 0.25)$ and the convergence is reached at $(1,1)$.

better but presents a quite irregular convergence toward the minimum. In particular, the algorithm goes beyond the minimum before heading back to it. The *l*-BFGS algorithm path is more regular; however, a zone of accumulation points is created near the minimum. Finally, the truncated Newton method proposes the most efficient path, taking long updates in the attraction valley before reaching the minimum, around which smaller steps are taken.

This example is completed by the analysis of the convergence curves of the four algorithms in terms of number of iterations and computational effort in Figure 6. Not surprisingly, the steepest-descent algorithm convergence is very slow compared with the three other methods. Although this does not appear in Figure 6, the convergence is reached only after several thousand iterations. On the other hand, the three other methods converge in less than 60 iterations. Among these three, the methods follow the hierarchy, which could be expected: the nonlinear conjugate gradient is the slowest (53 iterations), followed by the *l*-BFGS method (29 iterations); the most efficient one is the truncated Newton method (18 iterations).

However, this observation has to be mitigated by the analysis of the behavior of the methods in terms of computational effort. The latter is measured by the number of gradient computation required to reach convergence. In the case of the truncated-Newton method, the cost of a Hessian-vector multiplication is equivalent to the cost of a gradient computation. In terms of computational effort, the hierarchy between *l*-BFGS and the truncated Newton method is inverted. This is understandable because even if the truncated Newton method converges in less iterations, each iteration of this algorithm has a significantly higher cost related to the additional Hessian-vector products it has to perform.

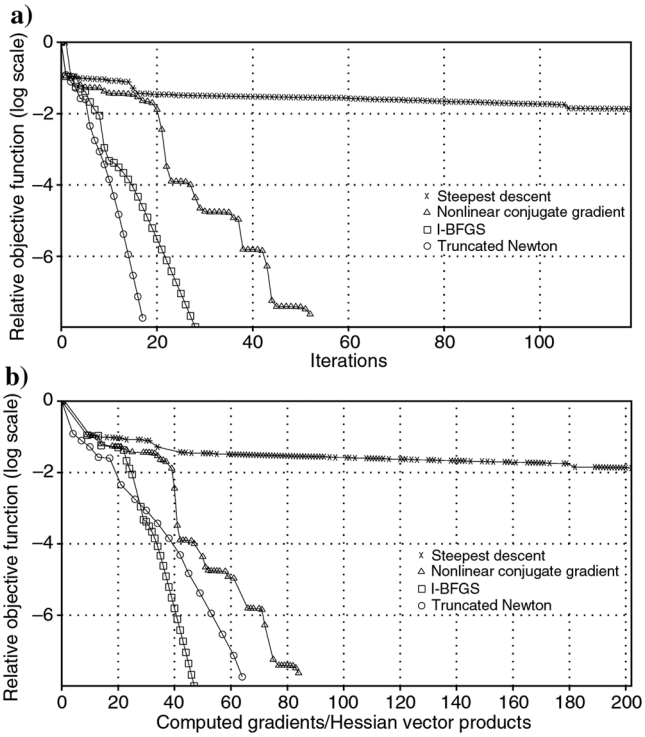

Figure 6. `Rosenbrock` function case study. Convergence rate of the four optimization methods in terms of (a) iterations and (b) computational cost in terms of the number of gradient evaluations. For the truncated Newton method, the computation cost of a Hessian-vector product is assimilated to the computation cost of a gradient.

The `Rosenbrock` case study is a good illustration of what can be expected from the different descent methods implemented in the SEISCOPE optimization toolbox. For difficult problems, the information on the local curvature of the objective function helps to guarantee a faster convergence speed. This information is embedded in the second-order derivatives of the objective function. The two methods accounting at best for this information are the *l*-BFGS method and the truncated Newton method. The latter has the capability to integrate the information on the Hessian with a great accuracy, which decreases the number of iterations required to reach the minimum, at the expense of an increased computational complexity. The *l*-BFGS method proposes the better trade-off between computational efficiency and accuracy of the Hessian estimation.

Full-waveform inversion on the Marmousi case study

Full-waveform inversion principle

FWI is a seismic imaging method based on the minimization between observed data and synthetic data computed through the numerical solution of a wave-propagation problem. The method allows us to retrieve high resolution quantitative estimates of subsurface parameters, such as P- and S-wave velocities, density, or anisotropy parameters. For this simple illustration, a 2D acoustic frequency-domain application with constant density is used. In this context, the wave propagation is described by the Helmholtz equation:

$$-\frac{\omega^2}{V_P^2}u - \Delta u = s, \tag{23}$$

where ω denotes the circular frequency, $V_P(x)$ is the P-wave velocity, $u(x,\omega)$ is the pressure wavefield, Δ is the Laplacian operator, and $s(x,\omega)$ is an explosive source term.

Given partial measurement of the pressure wavefield d_{obs} acquired at the surface, the FWI problem consists in the nonlinear least-squares minimization problem:

$$\min_{V_P} \frac{1}{2}\|d_{cal}(V_P) - d_{obs}\|^2, \tag{24}$$

where $\|.\|$ is the L^2-norm and $d_{cal}(V_P)$ is computed as

$$d_{cal}(V_P) = Ru(x,\omega;V_P). \tag{25}$$

In equation 25, $u(x,\omega;V_P)$ is the solution of the Helmholtz equation 23 for the P-wave velocity model V_P, and R is a restriction operator mapping this wavefield to the receiver location where the measurements are performed.

In the following numerical examples, the `TOY2DAC` code from the SEISCOPE group is used. This code is based on the solution of equation 23 through a fourth-order mixed staggered-grid finite-difference discretization (Hustedt et al., 2004). The solution of the Helmholtz equation in an infinite domain is simulated with perfectly matched layers (Bérenger, 1994). The associated linear system is solved by a lower upper triangular decomposition (LU) factorization performed using the massively parallel solver multifrontal massively parallel solver (MUMPS) (Amestoy et al., 2000; MUMPS-Team, 2011). The computation of the gradient and Hessian-vector products is per-

formed following first- and second-order adjoint methods. Using these techniques, the computation of these quantities is reduced to the solution of wave equation problems (equation 23) with different source terms (Plessix, 2006; Métivier et al., 2012, 2013). The TOY2DAC code is interfaced with the SEISCOPE optimization toolbox.

Comparison of the optimization routines on the Marmousi 2 benchmark model

The Marmousi 2 benchmark model and the initial model that is used are presented in Figure 7. The initial model is obtained after applying a Gaussian smoothing to the exact model with a 500 m correlation length. A fixed-spread acquisition system with 336 sources and receivers positioned at 50 m depth, from $x = 0.15$ to 16.9 km, equally spaced each 50 m, is used. The synthetic observed data set is constructed using six frequencies from 3 to 8 Hz, with 1 Hz sampling. No free-surface condition is implemented. The synthetic observed data thus do not contain any surface multiples. No multiscale strategy is used: The data corresponding to the six frequencies are inverted simultaneously. After discretization, the P-wave velocity model is described by approximately 100,000 discrete parameters, which yields a large-scale nonlinear optimization problem.

For FWI, the computational efficiency of the different optimization methods can be measured in terms of the number of gradients required to reach a certain level of accuracy. Indeed, a complexity analysis of the method reveals that the average complexity of one gradient computation is in $O(N^4)$ for 2D experiments ($O(N^6)$ for 3D experiments), where N is the average number of discrete points in one direction. This approximation is obtained assuming that the number of sources is in $O(N)$ (respectively $O(N^2)$ for the 3D case). In comparison, the complexity of the operations performed within the different optimization methods is in $O(N^2)$ (respectively $O(N^3)$ for the 3D case). Therefore, the computation associated with the optimization algorithm in itself is negligible compared with the computation cost associated with gradient computation. It should be mentioned here that in the context of adjoint strategies for computing Hessian-vector product, the computation cost of this operation is the same as the one for the gradient; therefore, this simplifies the comparison between first-order method and l-BFGS with the truncated Newton method.

The results obtained using the four different optimization methods available in the toolbox, steepest-descent, nonlinear conjugate gradient, and l-BFGS, truncated Newton, are presented in Figure 8. All the methods are combined with a diagonal preconditioner. To compute this preconditioner, the diagonal elements of the Hessian operator are approximated through the pseudo-Hessian approach promoted by Shin et al. (2001). The preconditioner is computed as the inverse of the diagonal matrix formed with these elements. A similar comparison of optimization methods in the context of FWI is performed by Castellanos et al. (2015) on different models.

The number of gradients stored for computing the l-BFGS approximation is set to $l = 10$. For the truncated Newton method,

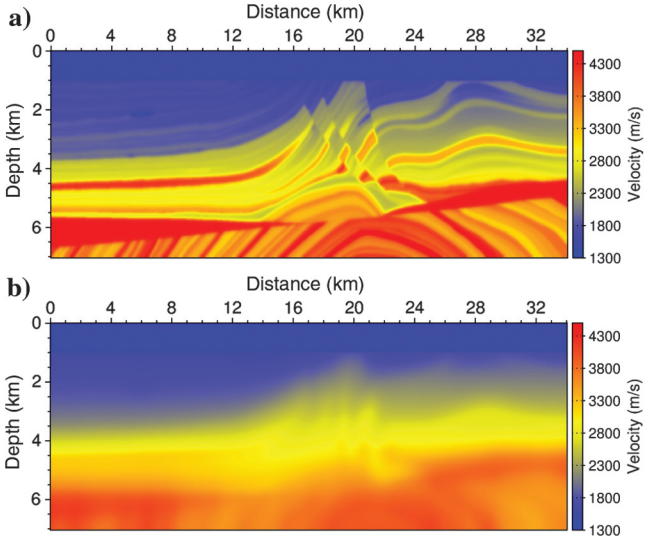

Figure 7. Marmousi 2 synthetic case study. (a) Exact model. (b) Initial model obtained after smoothing the exact model with a Gaussian filter. The correlation length for the Gaussian filter is set to 500 m. The number of discrete points is equal to 141×681, which yields a minimization problem involving approximately 96,000 unknowns.

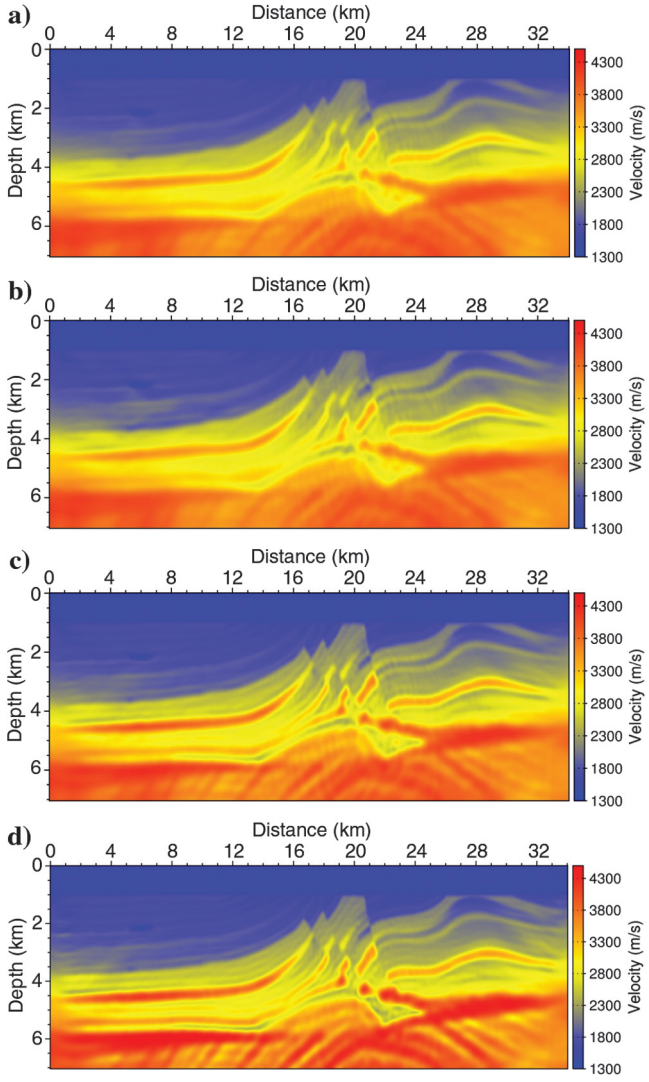

Figure 8. Marmousi case study. Models reconstructed by the four different optimization methods. (a) Steepest-descent, (b) nonlinear conjugate gradient, (c) l-BFGS, and (d) truncated Newton.

the maximum number of iterations for the incomplete solution of the Newton equations is set to 10. In practice, this bound is not reached: The stopping criterion from Eisenstat and Walker (1996) is satisfied at each nonlinear iteration in less than 10 iterations. The purpose of this experiment is to compare the different convergence rates of the methods; therefore, the stopping criterion that is used is based on the maximum number of nonlinear iterations to be performed, which is set to 20. The convergence curves in function of the number of iterations and the number of gradient estimations are presented in Figure 9.

As can be observed in Figure 9, the nonlinear conjugate gradient does not yield any improvement with respect to the steepest-descent method. This is not really a surprise, given the known erratic behavior of this method. In this situation, the method seems less efficient than the steepest-descent algorithm.

A second observation is the relatively good performance of the steepest-descent compared with the very slow convergence observed for the `Rosenbrock` test case for this method. Two reasons may be invoked to explain this observation. First, a preconditioner is used in the Marmousi 2 case, which improves significantly the convergence rate of this method. Second, the `Rosenbrock` case study is a pathological situation in which the steepest descent has the highest difficulty to converge. For the large-scale Marmousi 2 application, this seems to be no longer true.

As in the `Rosenbrock` case study, second-order methods outperform first-order methods. The truncated Newton method provides the fastest convergence rate in terms of nonlinear iterations. In terms of computational efficiency, the l-BFGS method provides the best performance.

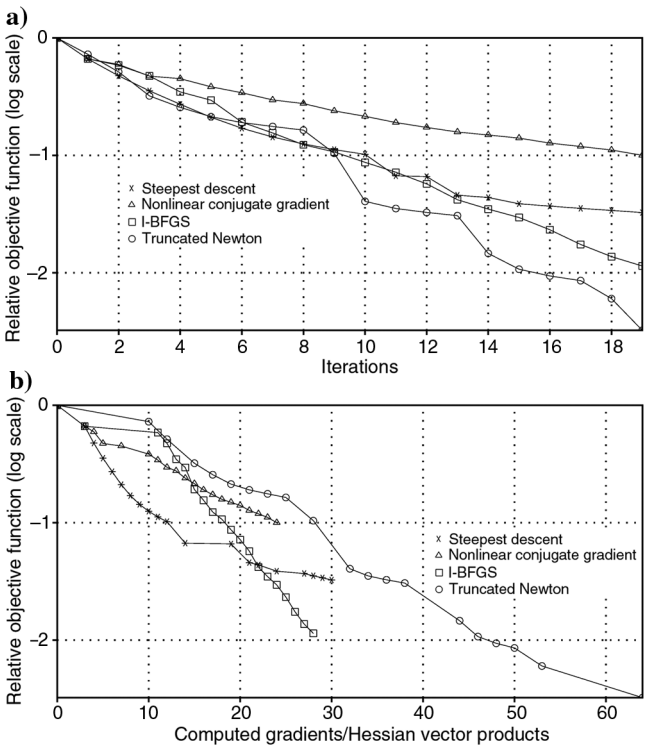

Figure 9. Marmousi case study. Convergence curves of the four different optimization methods in terms of (a) iterations and (b) computational cost in terms of the number of gradient evaluations. For the truncated Newton method, the computation cost of a Hessian-vector product is assimilated to the computation cost of a gradient. This is consistent with the implementation of second-order adjoint formulas for the computation of these quantities.

The analysis of the four P-wave estimations obtained after 20 iterations of each of the algorithm is in agreement with the objective function level attained by the four methods (Figure 8). The steepest-descent and nonlinear conjugate gradient estimations are comparable. The shallow structures are well recovered. The low-velocity anomalies accounting for the presence of gas located at ($z = 2$ and $x = 6$ km) and ($z = 2$ and $x = 21$ km) are efficiently reconstructed. However, the deeper structures below $z = 4.5$ km are not well delineated. Conversely, the l-BFGS and truncated Newton methods are better able to refocus the energy on this deeper part of the model. The deep reflectors are correctly reconstructed. This is particularly visible in the truncated Newton reconstruction. The inverse Hessian operator acts as an efficient deblurring filter in the context of FWI (Pratt et al., 1998).

Although the l-BFGS algorithm is more efficient than the truncated Newton method in terms of computational cost for this monoparameter FWI case study, the situation may change in the context of multiparameter FWI. In this context, nonnegligible trade-offs between different classes of parameters are expected (Operto et al., 2013). Removing these trade-offs require us to accurately account for the inverse Hessian operator at the first stages of the inversion (Métivier et al., 2014). Because the l-BFGS algorithm builds a progressive estimation of this operator along the iterations of the minimization loop, the estimation in the first iterations depends strongly on the prior information injected by the user. If this information is poor, as is often the case, it is expected that the trade-offs between parameter classes contaminate the solution at the early stages of the inversion. Removing these trade-offs in the next iterations is an extremely difficult task (see, for instance, the multiparameter FWI for ground penetrating radar data performed by Lavoué et al., 2014). In contrast, the truncated Newton method is based on an approximation of the inverse Hessian operator, whose accuracy does not depend on the convergence history. Therefore, it is expected that the truncated Newton method performs better than the l-BFGS algorithm in a multiparameter FWI context.

The comparison of the different minimization algorithms on the Marmousi case study should be concluded with the following comments: The settings of the experiments have been designed such that the initial model is close to the exact one. This could favor second-order algorithm over first-order algorithm because this is the configuration in which the superlinear convergence of second-order algorithms is more likely to be observed (in the vicinity of the minimum). In real case applications, the initial model can be poorer and the difference between first-order and second-order algorithm may be less obvious, at least in the early stages of the inversion. In any case, if the initial solution is very poor, all the four minimization strategies will fail to produce a reliable subsurface model estimation. What can be observed is an acceleration of the convergence rate when the solution approaches the minimum of the misfit function. The use of Tikhonov regularization also has an impact on the differences between first-order and second-order methods. This regularization technique, mandatory in case of noisy data for instance, results in adding a multiple of the identity to the Hessian operator. To see the impact of this regarding optimization method, one can consider the limit case, for which the Hessian operator tends toward the identity operator. In this case, first-order methods, relying on this approximation, become equivalent to second-order methods. Therefore, using strong Tikhonov regularization weights will have the

effect of reducing the differences between first- and second-order methods regarding their convergence rate.

CONCLUSION

The SEISCOPE optimization toolbox is a set of FORTRAN 90 routines for the solution of large-scale nonlinear minimization problems. This type of problems is ubiquitous in geophysics. The toolbox implements four different optimization schemes: the steepest-descent, the nonlinear conjugate gradient, the *l*-BFGS method, and the truncated Newton methods. All these routines are implemented with a line-search strategy ensuring the robustness of the methods: monotonic decrease of the objective function and guaranteed convergence toward the nearest local minimum. Bound constraints can also be activated.

The use of these routines within a computational code is completely uncoupled from the physical problem through a reverse communication protocol. Within this framework, the minimization of the objective function is brought back to the definition of a minimization loop controlled by the user in which the solver is called at each iteration. Depending on the return values of a communication flag, the user performs the action required by the solver at a given point, namely, computation of the objective function and its gradient, application of a preconditioner, and computation of a Hessian-vector product. This allows for flexible implementation and an easier use of second-order optimization methods.

The general structure of the minimization algorithms of the toolbox is based on a two-level reverse communication principle. The first level consists in implementing the communication between the user code and the minimization code. The second level is embedded in the minimization code itself and implements the communication between a line-search algorithm and the minimization code. The minimization code transfers the requirement from the line-search (computation of the objective function and its gradient) directly to the user. This imbrication of reverse communication levels is thus transparent for the user.

The example of use of the toolbox provided in the "Numerical results" section shows the benefit one can expect from second-order methods. The case of the Rosenbrock function is pathological; however, it illustrates how powerful the introduction of the information carried out by the second-order derivatives into the optimization can be. The Marmousi 2 case for FWI also illustrates the benefit of introducing an appropriate estimation of the inverse Hessian operator within the minimization to speed up the convergence of the algorithm and save considerable computation time.

Perspectives of development of the toolbox to adapt it to very large scale HPC applications is currently considered. In its actual implementation, the method requires to collect one model and one gradient on a single node of a supercomputer. For very large scale applications, such as the one involved in 3D FWI, these quantities are usually distributed, and this should require additional global communications at each iteration of the minimization loop. In some cases, the ability to store these quantities on a single node may even be questionable. Therefore, a completely distributed version of the toolbox may be derived to relax this requirement. One possible implementation is to externalize all the operations related to scalar products and vector additions involved in the minimization procedures through the reverse communication protocol. This could slightly complicate the interface of the toolbox with other computer codes, requiring more actions to be performed within the minimization loop. However, it would represent a very efficient way of performing the minimization on very large scale problems, for which all the vector quantities have to be distributed over the whole cluster.

On a longer term, the set of routines currently included in the toolbox could be extended to routines dedicated to the solution of constrained optimization problems. Based on the same architecture, sequential quadratic programming solvers could be designed for handling problems with not only bound constraints, but also linear and nonlinear equality and inequality constraints. These types of problems arise in geophysics as soon as regularization methods are considered, beyond simple additive techniques. Another longer term extension consists in integrating trust-region algorithms to propose an alternative to the line-search technique currently implemented. In particular, the combination of the truncated Newton method and the trust-region method, known as a the Steihaug algorithm, is reputed to benefit from better convergence properties than the line-search version of the truncated Newton method. These extensions may be the topic of future investigations.

ACKNOWLEDGEMENTS

This study was partially funded by the SEISCOPE consortium, sponsored by BP, CGG, Chevron, Exxon-Mobil, JGI, Petrobras, Saudi Aramco, Schlumberger, Shell, Sinopec, Statoil, Total, and Woodside. This study was granted access to the HPC resources of the Froggy platform of the CIMENT infrastructure, which is supported by the Rhône-Alpes region (GRANT CPER07_13 CIRA), the OSUG@2020 labex (reference ANR10 LABX56), and the Equip@Meso project (reference ANR-10-EQPX-29-01) of the programme Investissements d'Avenir supervised by the Agence Nationale pour la Recherche, and the HPC resources of CINES/IDRIS under the allocation 046091 made by GENCI. The authors would like to thank all the staff of the SEISCOPE Consortium research group for their active and constant support. Special thanks are due to S. Operto for suggesting to us this article and his (always) careful proofreading. Special thanks also go to J. Dellinger, P. Thierry, E. Diaz, and two anonymous reviewers for their suggestions, which really helped us to improve our work.

APPENDIX A

PRECONDITIONED L-BFGS ALGORITHM

The *l*-BFGS algorithm is based on the computation of the descent direction Δx_k following equation 8. In equation 8, Q_k is defined by

$$\begin{aligned} Q_k = {} & (V_{k-1}^T \dots V_{k-l}^T) Q_k^0 (V_{k-l} \dots V_{k-1}) \\ & + \rho_{k-l} (V_{k-1}^T \dots V_{k-l+1}^T) s_{k-l} s_{k-l}^T (V_{k-l+1} \dots V_{k-1}) \\ & + \rho_{k-l+1} (V_{k-1}^T \dots V_{k-l+2}^T) s_{k-l+1} s_{k-l+1}^T (V_{k-l+2} \dots V_{k-1}) \\ & + \dots \\ & + \rho_{k-1} s_{k-1} s_{k-1}^T, \end{aligned} \tag{A-1}$$

where the pairs s_k, y_k are

$$s_k = x_{k+1} - x_k, \qquad y_k = \nabla f(x_{k+1}) - \nabla f(x_k), \tag{A-2}$$

the scalar ρ_k are

Algorithm 1. Two-loop recursion algorithm.

Data: $\rho_i, s_i, y_i, i = k-l, \ldots, k-1, H_k^0, \nabla f(x_k)$
Result: $\Delta x_k = -Q_k \nabla f(x_k)$
$q = -\nabla f(x_k)$;
for $i = k-1, \ldots, k-l$ **do**
  $\alpha_i = \rho_i s_i^T \Delta x_k$;
  $q = q - \alpha_i y_i$;
End
$\Delta x_k = H_k^0 q$;
for $i = k-l, \ldots, k-1$ **do**
  $\beta = \rho_i y_i^T \Delta x_k$;
  $\Delta x_k = \Delta x_k + (\alpha_i - \beta_i) s_i$;
End

$$\rho_k = \frac{1}{y_k^T s_k}, \tag{A-3}$$

and the matrices V_k are defined by

$$V_k = I - \rho_k y_k s_k^T. \tag{A-4}$$

These formulas are directly extracted from Nocedal and Wright (2006). As is also explained in this reference textbook, the computation of the quantity $\Delta x_k = -Q_k \nabla f(x_k)$, can be performed through the double-recursion algorithm presented below.

In Algorithm 1, the matrix H_k^0 is an estimation of the inverse Hessian matrix at the kth iteration of the l-BFGS minimization. In most applications, this estimation is not updated throughout the l-BFGS iterations, and kept equal to H_0^0. However, nothing prevents us from performing this update. In practice, in the FWI application presented in this study, this strategy reveals itself to be efficient. The inverse Hessian estimation that is used is a diagonal approximation based on the pseudo-Hessian strategy (Shin et al., 2001). The computation cost of this approximation is cheap, and it therefore can be recalculated at each iteration. The implementation that is used in the toolbox allows for these updates to be performed. When the routine `PLBFGS` is used, the preconditioning operation requested by the communicator flag set to "`PREC`" corresponds to the multiplication $\Delta x_k = H_k^0 q$. The user may thus update the matrix H_k^0 at each iteration k before performing this multiplication. Surprisingly, this possibility is rarely applied (to the best of our knowledge), although it is explicitly specified in Nocedal and Wright (2006).

REFERENCES

AGL, 2015, http://www.agl.uh.edu/downloads/downloads.htm, accessed 23 September 2015

Amestoy, P., I. S. Duff, and J. Y. L'Excellent, 2000, Multifrontal parallel distributed symmetric and unsymmetric solvers: Computer Methods in Applied Mechanics and Engineering, **184**, 501–520, doi: 10.1016/S0045-7825(99)00242-X.

Baumstein, A., 2013, POCS-based geophysical constraints in multi-parameter full wavefield inversion: Presented at the 75th Annual International Conference and Exhibition, EAGE.

Bérenger, J.-P., 1994, A perfectly matched layer for absorption of electromagnetic waves: Journal of Computational Physics, **114**, 185–200, doi: 10.1006/jcph.1994.1159.

Bonnans, J. F., J. C. Gilbert, C. Lemaréchal, and C. A. Sagastizábal, 2006, Numerical optimization: Theoretical and practical aspects, Universitext: Springer.

Byrd, R. H., P. Lu, and J. Nocedal, 1995, A limited memory algorithm for bound constrained optimization: SIAM Journal on Scientific and Statistical Computing, **16**, 1190–1208, doi: 10.1137/0916069.

Castellanos, C., L. Métivier, S. Operto, R. Brossier, and J. Virieux, 2015, Fast full waveform inversion with source encoding and second-order optimization methods: Geophysical Journal International, **200**, 718–742, doi: 10.1093/gji/ggu427.

Dai, Y., and Y. Yuan, 1999, A nonlinear conjugate gradient method with a strong global convergence property: SIAM Journal on Optimization, **10**, 177–182, doi: 10.1137/S1052623497318992.

Dongarra, J., V. Eijkhout, and A. Kalhan, 1995, Reverse communication interface for linear algebra templates for iterative methods: Technical report, University of Tennessee.

Eisenstat, S. C., and H. F. Walker, 1996, Choosing the forcing terms in an inexact Newton method: SIAM Journal on Scientific Computing, **17**, 16–32, doi: 10.1137/0917003.

Fletcher, R., and C. M. Reeves, 1964, Function minimization by conjugate gradient: Computer Journal, **7**, 149–154, doi: 10.1093/comjnl/7.2.149.

Gould, N. I. M., S. Lucidi, M. Roma, and P. Toint, 1999, Solving the trust-region subproblem using the lanczos method: SIAM Journal on Optimization, **9**, 504–525, doi: 10.1137/S1052623497322735.

Hestenes, M. R., and E. Stiefel, 1952, Methods of conjugate gradient for solving linear systems: Journal of Research of the National Bureau of Standards, **49**, 409–436, doi: 10.6028/jres.049.044.

Hustedt, B., S. Operto, and J. Virieux, 2004, Mixed-grid and staggered-grid finite-difference methods for frequency-domain acoustic wave modeling: Geophysical Journal International, **157**, 1269–1296, doi: 10.1111/j.1365-246X.2004.02289.x.

Lambaré, G., J. Virieux, R. Madariaga, and S. Jin, 1992, Iterative asymptotic inversion in the acoustic approximation: Geophysics, **57**, 1138–1154, doi: 10.1190/1.1443328.

Lavoué, F., R. Brossier, L. Métivier, S. Garambois, and J. Virieux, 2014, Two-dimensional permittivity and conductivity imaging by full waveform inversion of multioffset GPR data: A frequency-domain quasi-Newton approach: Geophysical Journal International, **197**, 248–268, doi: 10.1093/gji/ggt528.

Martin, G. S., R. Wiley, and K. J. Marfurt, 2006, Marmousi2: An elastic upgrade for Marmousi: The Leading Edge, **25**, 156–166, doi: 10.1190/1.2172306.

Métivier, L., F. Bretaudeau, R. Brossier, S. Operto, and J. Virieux, 2014, Full waveform inversion and the truncated Newton method: Quantitative imaging of complex subsurface structures: Geophysical Prospecting, **62**, 1353–1375, doi: 10.1111/1365-2478.12136.

Métivier, L., R. Brossier, J. Virieux, and S. Operto, 2012, Toward Gauss-Newton and exact Newton optimization for full waveform inversion: Presented at the 74th Annual International Conference and Exhibition, EAGE.

Métivier, L., R. Brossier, J. Virieux, and S. Operto, 2013, Full waveform inversion and the truncated Newton method: SIAM Journal on Scientific Computing, **35**, B401–B437, doi: 10.1137/120877854.

MUMPS-Team, 2011, MUMPS — Multifrontal massively parallel solver users' guide — Version 4.10.0 (May 10, 2011): ENSEEIHT-ENS Lyon.

Nash, S. G., 2000, A survey of truncated Newton methods: Journal of Computational and Applied Mathematics, **124**, 45–59, doi: 10.1016/S0377-0427(00)00426-X.

Nash, S. G., and J. Nocedal, 1991, A numerical study of the limited memory BFGS method and the truncated-Newton method for large scale optimization: SIAM Journal on Optimization, **1**, 358–372, doi: 10.1137/0801023.

Nemeth, T., C. Wu, and G. T. Schuster, 1999, Least-squares migration of incomplete reflection data: Geophysics, **64**, 208–221, doi: 10.1190/1.1444517.

Nocedal, J., 1980, Updating Quasi-Newton matrices with limited storage: Mathematics of Computation, **35**, 773–773, doi: 10.1090/S0025-5718-1980-0572855-7.

Nocedal, J., and S. J. Wright, 2006, Numerical optimization, 2nd ed.: Springer.

Nolet, G., 1987, Seismic tomography with applications in global seismology and exploration geophysics: D. Reidel Publishing Company.

Operto, S., R. Brossier, Y. Gholami, L. Métivier, V. Prieux, A. Ribodetti, and J. Virieux, 2013, A guided tour of multiparameter full waveform inversion for multicomponent data: From theory to practice: The Leading Edge, **32**, 1040–1054, doi: 10.1190/tle32091040.1.

Plessix, R. E., 2006, A review of the adjoint-state method for computing the gradient of a functional with geophysical applications: Geophysical Journal International, **167**, 495–503, doi: 10.1111/j.1365-246X.2006.02978.x.

Polak, E., and G. Ribière, 1969, Note sur la convergence de méthodes de directions conjuguées: Revue Française d'Informatique et de Recherche Opérationnelle, **16**, 35–43.

Pratt, R. G., C. Shin, and G. J. Hicks, 1998, Gauss-Newton and full Newton methods in frequency-space seismic waveform inversion: Geophysical

Journal International, **133**, 341–362, doi: 10.1046/j.1365-246X.1998.00498.x.
Sen, M. K., and P. L. Stoffa, 1995, Global optimization methods in geophysical inversion: Elsevier Science Publishing Co.
Seiscope, 2015, http://seiscope2.osug.fr/TOY2DAC,82?lang=en, accessed 23 September 2015.
Shin, C., S. Jang, and D. J. Min, 2001, Improved amplitude preservation for prestack depth migration by inverse scattering theory: Geophysical Prospecting, **49**, 592–606, doi: 10.1046/j.1365-2478.2001.00279.x.
Spall, J. C., 2003, Introduction to stochastic search and optimization: Estimation, simulation and control: Wiley-Interscience Series in Discrete Mathematics and Optimization.
Steihaug, T., 1983, The conjugate gradient method and trust regions in large scale optimization: SIAM Journal on Numerical Analysis, **20**, 626–637, doi: 10.1137/0720042.
Virieux, J., and S. Operto, 2009, An overview of full waveform inversion in exploration geophysics: Geophysics, **74**, no. 6, WCC1–WCC26, doi: 10.1190/1.3238367.