SEISCOPE OPTIMIZATION TOOLBOX MANUAL

Ludovic Métivier
ludovic.metivier@ujf-grenoble.fr

code version 1.0 - SVN revision 3873 - July 2014

SEISCOPE Consortium
`http://seiscope2.osug.fr`

Legal statement

Acknowledgments

The SEISCOPE OPTIMIZATION TOOLBOX codes have been developed in the framework of the SEISCOPE and SEISCOPE II consortia and we thank the sponsors of these projects. We also thank the French National Center for Scientific Research (CNRS) for his support. Access to the high performance computing facilities of the meso-center CIMENT (Univ. Grenoble Alpes, Fr, https://ciment.ujf-grenoble.fr/) provided the required computer resources to develop this package

Conditions of use

The SEISCOPE OPTIMIZATION TOOLBOX code is provided open-source (see Legal Statement). Please refer to the two following articles in any study or publications for which this code has been used

- Full Waveform Inversion and the truncated Newton method: Quantitative imaging of complex subsurface structures, 2014, Geophysical Prospecting, L. Métivier, R. Brossier, S. Operto, J. Virieux, DOI: 10.1111/1365-2478.12136, Métivier et al. (2014)
- Full Waveform Inversion and the Truncated Newton Method, L.Métivier, R.Brossier, J.Virieux, S.Operto, 2013, SIAM J. Sci. Comput, 35(2), B401-B437, Métivier et al. (2013)

Contents

1 Introduction

The SEISCOPE OPTIMIZATION TOOLBOX is a library of optimization routines developed in `FORTRAN 90` for solving unconstrained and bound constrained nonlinear large-scale minimization problems. Six optimization methods are implemented.

1. The (preconditioned) steepest descent.
2. The (preconditioned) nonlinear conjugate gradient.
3. The l-BFGS method.
4. The preconditioned l-BFGS method.
5. The truncated Newton method.
6. The preconditioned truncated Newton method.

The library is self-consistent: no other existing `FORTRAN` libraries are needed to use the code. All the routines of the SEISCOPE OPTIMIZATION TOOLBOX are implemented in a reverse communication framework to facilitate their use.

This manual is organized as follows:

- In Section 2, we give a quick overview of the different optimization methods available in the SEISCOPE OPTIMIZATION TOOLBOX. References to detailed presentation of the algorithms which are implemented are given.
- In Section 3, we present how to install the SEISCOPE OPTIMIZATION TOOLBOX. For each of the optimization routines, a simple test case is provided. The corresponding programs should be used as templates for using the SEISCOPE OPTIMIZATION TOOLBOX routines.
- In Section 4, we present in details how to use each of the optimization routines.
- In Section 5, additional information on some technical points are given.

The information contained in the manual is redundant: there is no many differences in the use of the different routines. However, we prefer repeating the information. The user can directly read the part corresponding to the routine he wants to use without having to real all the documentation about the other routines.

2 An overview of the routines in the toolbox

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

$$\min_{x\in\Omega} f(x) \tag{2.1}$$

where

$$\Omega = \prod_{i=1}^{n} [a_i, b_i] \subset \mathcal{R}^n \quad n \in \mathcal{N} \tag{2.2}$$

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

$$x_{k+1} = x_k + \alpha_k \Delta x_k, \tag{2.3}$$

where

- α_k is a steplength;
- Δx_k is a descent direction.

The recurrence (2.3) is repeated until convergence is reached (in a certain sense). The steplength α_k is computed through a linesearch process which is the same for all the routines in the TOOLBOX. This linesearch process satisfies the Wolfe conditions: this ensures that from an arbitrary initial guess, convergence toward a **local** minimum will be obtained, 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 linesearch process.

The computation of the descent direction differs from one routine to the other. The different routines which are implemented in the SEISCOPE OPTIMIZATION TOOLBOX are presented in the following.

2.1 Preconditioned steepest descent: PSTD

The preconditioned steepest descent uses the following descent direction

$$\Delta x_k = -P_k \nabla f(x_k), \tag{2.4}$$

where

- $\nabla f(x_k)$ is the gradient of the function $f(x_k)$ at point x_k;
- P_k is an arbitrary preconditioning matrix.

Following the reverse communication implementation ot the SEISCOPE OPTIMIZATION TOOLBOX, the information the user has to provide to the solver for using the PSTD routine is thus

- the cost function $f(x_k)$ for a given x_k;
- the gradient of the cost function $\nabla f(x_k)$ for a given x_k;
- the gradient of the cost function multiplied by the preconditioner $P_k \nabla f(x_k)$ for a given x_k where P_k is the preconditioner

Note that if the user does not have any preconditioner at hand, the use of the identity is possible. The method thus reduces to a standard steepest-descent method.

2.2 Preconditioned nonlinear conjugate gradient: PNLCG

The preconditioned nonlinear conjugate gradient method uses the following descent direction

$$\Delta x_k = -P_k \nabla f(x_k) + \beta_k \Delta x_{k-1}, \tag{2.5}$$

where

- $\nabla f(x_k)$ is the gradient of the function $f(x_k)$ at point x_k;
- P_k is an arbitrary preconditioning matrix;
- β_k is computed following the formula of Dai and Yuan (1999).

Following the reverse communication implementation of the SEISCOPE OPTIMIZATION TOOLBOX, the information the user has to provide to the solver for using the PNLCG routine is thus

- the cost function $f(x_k)$ for a given x_k;
- the gradient of the cost function $\nabla f(x_k)$ for a given x_k;
- the gradient of the cost function multiplied by the preconditioner $P_k \nabla f(x_k)$ for a given x_k where P_k is the preconditioner

Note again that if the user does not have preconditioner at hand, the use of the identity is possible. The method thus reduce to a standard nonlinear conjugate gradient method.

2.3 Quasi-Newton l-BFGS method: LBFGS

The l-BFGS method uses the following descent direction

$$\Delta x_k = -Q_k \nabla f(x_k), \tag{2.6}$$

where

- $\nabla f(x_k)$ is the gradient of the function $f(x_k)$ at point x_k;
- Q_k is the l-BFGS approximation of the inverse Hessian operator $H(x_k)^{-1} = \nabla^2 f(x_k)^{-1}$ (more details on this approximation can be found in Byrd et al. (1995); Nocedal and Wright (2006)).

Following the reverse communication implementation ot the SEISCOPE OPTIMIZATION TOOLBOX, the information the user has to provide to the LBFGS routine is thus

- the cost function $f(x_k)$ for a given x_k;
- the gradient of the cost function $\nabla f(x_k)$ for a given x_k.

The l-BFGS approximation Q_k is directly determined by the LBFGS routine. No specific action of the user is requested to compute it.

2.4 Quasi-Newton preconditioned l-BFGS method: PLBFGS

The preconditioned l-BFGS method uses the following descent direction

$$\Delta x_k = -\tilde{Q}_k \nabla f(x_k), \tag{2.7}$$

where

- $\nabla f(x_k)$ is the gradient of the function $f(x_k)$ at point x_k;
- $\tilde{Q}_k$ is the l-BFGS approximation of the inverse Hessian operator $H(x_k)^{-1} = \nabla^2 f(x_k)^{-1}$ computed from an initial estimation P_k of $H(x_k)^{-1}$ (more details on this approximation can be found in Byrd et al. (1995); Nocedal and Wright (2006)).

Following the reverse communication implementation ot the SEISCOPE OPTIMIZATION TOOLBOX, the information the user has to provide to the LBFGS routine is thus

- the cost function $f(x_k)$ for a given x_k;
- the gradient of the cost function $\nabla f(x_k)$ for a given x_k;
- the preconditioned gradient of the cost function $P_0 \nabla f(x_0)$ at the first iteration;
- the multiplication of a given vector v by a preconditioning matrix P_k provided by the user: $P_k v$.

The only difference with the LBFGS routine is that an additional information P_k on the inverse Hessian approximation is inserted in the computation of Q_k. For the user, this amounts to a preconditioning operation, since this information is taken into account by multiplying a vector by P_k. *This can drastically enhance the convergence of the conventional l-BFGS algorithm.*

2.5 Truncated Newton method: TRN

The truncated Newton method computes an approximate solution of the linear system

$$H(x_k)\Delta x_k = -\nabla f(x_k), \tag{2.8}$$

where

- $\nabla f(x_k)$ is the gradient of the function $f(x_k)$ at point x_k;
- $H(x_k)$ is the Hessian operator $H(x_k) = \nabla^2 f(x_k)$.

This approximate solution of the linear system is computed through a matrix-free conjugate gradient algorithm. The stopping criterion for this system is

$$\|H(x_k)\Delta x_k + \nabla f(x_k)\| \leq \eta_k \|\nabla f(x_k)\|; \tag{2.9}$$

where η_k is a forcing term which depends on the gradient current and previous value (see Eisenstat and Walker (1994); Métivier et al. (2013) for more details).

Following the reverse communication implementation of the SEISCOPE OPTIMIZATION TOOLBOX, the information the user has to provide to the TRN routine is thus

- the cost function $f(x_k)$ for a given x_k;
- the gradient of the cost function $\nabla f(x_k)$ for a given x_k;
- the multiplication of a given vector v by the Hessian matrix $H(x_k)$: $H(x_k)v$.

2.6 Preconditioned truncated Newton method: PTRN

The preconditioned truncated Newton method computes an inexact solution of the linear system

$$P_k H(x_k)\Delta x_k = -P_k \nabla f(x_k), \tag{2.10}$$

where

- $\nabla f(x_k)$ is the gradient of the function $f(x_k)$ at point x_k;
- $H(x_k)$ is the Hessian operator $H(x_k) = \nabla^2 f(x_k)$;
- P_k is a preconditioning matrix.

This inexact solution of the linear system is computed through a matrix-free conjugate gradient algorithm. The stopping criterion for this system is

$$\|H(x_k)\Delta x_k + \nabla f(x_k)\| \leq \eta_k \|\nabla f(x_k)\|, \tag{2.11}$$

where η_k is a forcing term which depends on the gradient current and previous value (see Eisenstat and Walker (1994); Métivier et al. (2013) for more details).

Following the reverse communication implementation ot the SEISCOPE OPTIMIZATION TOOLBOX, the information the user has to provide to the PTRN routine is thus

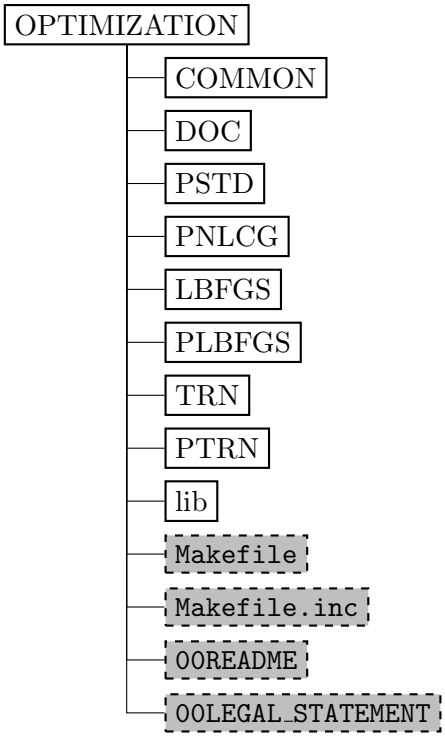
- the cost function $f(x_k)$ for a given x_k;
- the gradient of the cost function $\nabla f(x_k)$ for a given x_k;
- the multiplication of a given vector v by the preconditioner P_k: $P_k v$;
- the multiplication of a given vector v by the Hessian matrix $H(x_k)$: $H(x_k)v$.

3 Installation

To install the SEISCOPE OPTIMIZATION TOOLBOX, the user first has to decompress the file `SEISCOPE_OPTIMIZATION_TOOLBOX.tgz`. This is achieved through the command

```
tar -xvzf SEISCOPE_OPTIMIZATION_TOOLBOX.tgz
```

This command will create the following directories



The SEISCOPE OPTIMIZATION TOOLBOX is used as a static library. This means that the set of routines are gathered in a file `*.a` after the compilation. This library has to be linked by the program calling the routines from the SEISCOPE OPTIMIZATION TOOLBOX. Examples are provided in the sequel for generating small test programs.

3.1 Compilation

For compiling the library and generate the file `libSEISCOPE_OPTIM.a`, the user first has to open the file `Makefile.inc` and define which compiler should be used. This is done by editing the first lines of `Makefile.inc`. The default compiler is `ifort`:

```
FC = ifort
```

Different compilation options can be chosen by modifying the macro `OPTF`. The default option is

`FLAG = -O3 -assume byterecl`.

Once this is done, the user simply has to type in the terminal the command

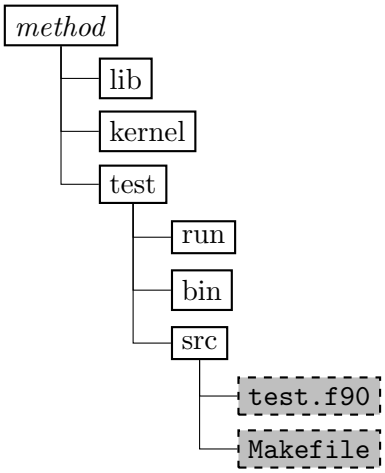
```
make lib
```

This will generate the file `libSEISCOPE_OPTIM.a` in the directory `./lib`. It is possible to remove all compiled files (objects and library files) by typing the command

```
make clean
```

3.2 Compiling and running the test programs

Once the library has been compiled, the test programs can be compiled and executed. For each optimization method, the test directory is organized as follows:



The source code of the test program is in the file `test.f90`. The test consists in the minimization of the 2D Rosenbrock function.

$$f(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2 \tag{3.1}$$

This function is famous in the optimization community as an example of a non-convex function with a global minimum in a narrow flat shaped valley. The convergence to this global minimum is difficult. The source code of the Rosenbrock function is in

`OPTIMIZATION/COMMON/test/rosenbrock.f90`.

The function `Rosenbrock(x,fcost,grad)` computes the cost function and its gradient at point `x` and return their values in `fcost` and `grad` respectively.

The function `Rosenbrock_Hess(x,d,Hd)` computes the product of the vector `d` by the Hessian operator $H(x)$ and return the value in `Hd`.

To compile the test program, the `Makefile` in the directory `OPTIMIZATION/`*`method`*`/test/src` has to be edited. As for the compilation of the library, the compiler as to be defined by editing

the first line of the `Makefile`. The default one is `mpif90`:

```
CXX = mpi90
```

Again, different compilation options can be chosen by modifying the macro `FLAGS`. The default option is

```
FLAG = -O3 -assume byterecl -warn noalign.
```

Once this is done, the compilation is done by typing

```
make
```

in the directory `OPTIMIZATION/`*`method`*`/test/src`. This will automatically create the test object files and link them to the library. The result of the compilation is the executable file

```
OPTIMIZATION/method/test/bin/test.bin
```

The compiled files (objects and executable files) can be removed by typing the command

```
make clean
```

in the directory `OPTIMIZATION/`*`method`*`/test/src`. To run the executable file, the use has to move to the directory `OPTIMIZATION/`*`method`*`/test/run` and type the command

```
../bin/test_bin
```

The output on the console should be (for the PSTD algorithm)

```
END OF TEST
FINAL iterate is :  1.000495 1.000981
See the convergence history in iterate_ST.dat
```

4 How to use the optimization routines?

All the routines of the SEISCOPE OPTIMIZATION TOOLBOX are implemented in a reverse communication form. The function $f(x)$ is minimized in a loop in which at each iteration, the solver from the SEISCOPE OPTIMIZATION TOOLBOX is called. In return, a communication FLAG tells the user the action he has to perform. These actions can be listed as:

- compute the cost function;
- compute the gradient;
- apply the preconditioner;
- apply the Hessian operator.

To use the optimization routines of the SEISCOPE OPTIMIZATION TOOLBOX, *we recommend the user to use the test files in the directory of each optimization routines as template files.* We describe in details these templates files in the following subsections.

4.1 Preconditioned Steepest Descent: PSTD

4.1.1 Variables declaration

The first step for using the PSTD method is to declare the inputs and outputs of the function.

1. Include the header file optim.h to declare the data structure optim which will contain most of the information required by PSTD.
2. Declare the integer n, dimension of the problem (2.1).
3. Declare the real `fcost`, the cost function f computed at x.
4. Declare the vector `x`, the unknown $x \in \mathcal{R}^n$ of the minimization problem (2.1).
5. Declare the vector `grad`, which corresponds to the gradient of the cost function f at x: $\nabla f(x) \in \mathcal{R}^n$.
6. Declare the vector `grad_preco`, which corresponds to the preconditioned gradient of the cost function f at x: $P\nabla f(x) \in \mathcal{R}^n$.
7. Declare the data structure optim.
8. Declare the chain of character of length 4 FLAG. This is the flag for the reverse communication between the PSTD routine and the user.

```
implicit none
include 'optim_type.h'

integer :: n                                  ! dimension of the problem
real :: fcost                                 ! cost function value
real,dimension(:),allocatable :: x            ! current point
real,dimension(:),allocatable :: grad         ! current gradient
real,dimension(:),allocatable :: grad_preco   ! preconditioned gradient
type(optim_type) :: optim                     ! data structure for the optimizer
character*4 :: FLAG                           ! communication FLAG
```

Declaration of the FORTRAN variables.

4.1.2 Initialization

The second step consists in initializing the problem.

1. Set the dimension n of the problem, n should be chosen such that $n \geq 1$.

2. Initialize the communication flag to 'INIT'.

3. Set the maximum number of nonlinear iteration that can be performed `optim%niter_max`.

4. Set the tolerance parameter for the stopping criterion `optim%conv`. The stopping criterion which is implemented by default is

$$(f(x)/f(x_0) < \texttt{optim\%conv}) \text{ OR } (\texttt{optim\%niter} \geq \texttt{optim\%niter\_max}). \tag{4.1}$$

   This means that the flag 'CONV' will be returned to the user when this condition is met. However, the user can define his own convergence criterion, as he defines himself the convergence loop (see next section).

5. Set the flag for printing output: if `optim%print_flag` is set to 1 the output files containing information on the convergence history are created. No output files are created otherwise.

6. Set the flag for using/not using bound constraints: if `optim%bound` is set to 1, bound constraints are used. If `optim%bound` is set to 0, no bound constraints are imposed.

7. If bound constraints have been activated, then the user must set additional variables. First, allocate the vectors `optim%ub` and `optim%lb` for respectively "upper bounds" and "lower bounds". The size of this vectors has to be equal to the dimension of the optimization problem n. Then set `optim%ub` and `optim%lb` with the corresponding bound constraints values. Each component i of these vectors define upper and lower bounds for the component x_i of the unknown. Finally, set the tolerance `optim%threshold`. This value gives the tolerance with which the bound constraints are satisfied. In practice, we enforce the condition

$$\texttt{optim\%lb}_i + \texttt{optim\%threshold} \leq x_i \leq \texttt{optim\%ub}_i - \texttt{optim\%threshold} \tag{4.2}$$

8. Set the level of of information in the output file `iterate_ST.dat`: if `optim%debug` is set to `true` then information concerning the linesearch process will be printed, otherwise, if it is set to `false`, the file `iterate_ST.dat` will contain only the convergence history (see 5.2.5 for more details).

9. Define the initial guess: the unknown `x` has to be allocated and initialized to a specific value.

10. VERY IMPORTANT: compute the cost function and the gradient corresponding to the initial guess and store the result in `fcost` and `grad`.

11. VERY IMPORTANT: multiply the gradient by the preconditioner and store the result in `grad_preco`. Note that if no preconditioner is available, you simply have to copy `grad` in `grad_preco`. This means that the preconditioner is identity. In this case the PSTD method is a standard steepest descent method (without preconditioning).

It is very important that ***fcost****,* ***grad*** *and* ***grad_preco*** *are initialized to the values corresponding to the initial guess* ***x*** *on the first call to the solver PSTD.*

```
n=2                              ! dimension
FLAG='INIT'                      ! first flag
optim%niter_max=10000            ! maximum iteration number
optim%conv=1e-8                  ! tolerance for the stopping criterion
optim%debug=.false.              ! level of details for output files
optim%pring_flag=1               ! print information
optim%bound=1                    ! activation of bound constraints
allocate(optim%ub(n))            ! allocate upper bound
allocate(optim%lb(n))            ! allocate lower bound
ub(:)=40.                        ! set upper bound
lb(:)=-40.                       ! set lower bound
optim%threshold=1e-2             ! set tolerance for bound constraints

allocate(x(n),grad(n),grad_preco(n)) ! allocation
x(1)=1.5                         ! set initial point
x(2)=1.5

call Rosenbrock(x,fcost,grad)    ! initial cost and gradient
grad_preco(:)=grad(:)            ! no preconditioning
```

Initialization.

4.1.3 Minimization within the reverse communication loop

The third step consists in performing the minimization within the reverse communication loop.

1. Define the `while` loop. In the chosen example, the loop is terminated when the convergence criterion is satisfied (or the maximum number of iteration has been reached), in this case the communication flag is 'CONV'. The loop is also terminated when the flag returned by the PSTD solver is 'FAIL', which indicates that the linesearch process has failed in finding a suitable steplength in the current descent direction.

2. At each iteration of the while loop, call the PSTD routine. On first call, the flag is set to 'INIT', the unknown `x` is initialized to the initial guess x_0, and the variables `fcost`, `grad` and `grad_preco` contain respectively $f(x_0)$, $\nabla f(x_0)$ and $P_0 \nabla f(x_0)$.

3. On return of the call to the PSTD routine, if the communication flag is 'GRAD' then the value of x has been modified. Compute the cost function $f(x)$ and the gradient $\nabla f(x)$ at this new point in the variables `fcost` and `grad`. If the user wants, he can also apply a preconditioner to the gradient value. In this case, the value $P_k \nabla f(x)$ is stored in `grad_preco`. Note that the preconditioner can change throughout the iterations.

```
do while ((FLAG.ne.'CONV').and.(FLAG.ne.'FAIL'))
   call PSTD(n,x,fcost,grad,grad_preco,optim,FLAG)
   if(FLAG.eq.'GRAD') then
      !compute cost and gradient at point x
      call Rosenbrock(x,fcost,grad)
      !no preconditioning
      grad_preco(:)=grad(:)
   endif
enddo
```

Reverse communication loop.

4.1.4 End of the loop and output file

At the end of the reverse communication loop, either convergence has been reached and the communication flag is 'CONV', or the linesearch has failed and the communication flag is 'FAIL'. In both cases, the vector `x` contains the last value of the minimization sequence (2.3), the best approximation to the solution of the minimization problem that can be found using PSTD.

The convergence history is written in the file `iterate_ST.dat`. This file contains a remainder of the optimization settings: convergence criterion and maximum number of iteration. He also presents the initial cost without normalization and the initial norm of the gradient. Then, it presents on 7 columns the convergence history.

- Column 1 is for the nonlinear iteration number.
- Column 2 is for the non normalized cost function value.
- Column 3 is for the norm of the gradient.
- Column 4 is for the relative cost function value (normalized by the initial value, on the first iteration this value is then always equal to 1).
- Column 5 is for the size of the steplength taken.
- Column 6 is for the number of linesearch iteration for determining the steplength.
- Column 7 is for the total number of gradient computation.

```
*********************************************************************
              STEEEPEST DESCENT ALGORITHM
*********************************************************************
     Convergence criterion   :    1.00E-08
     Niter_max               :    10000
     Initial cost is         :    5.65E+01
     Initial norm_grad is    :    4.75E+02
*********************************************************************
   Niter      fk          ||gk||       fk/f0        alpha        nls      ngrad
     0     5.65E+01     4.75E+02     1.00E+00     1.00E+00       0          0
     1     2.74E+01     2.45E+02     4.86E-01     9.77E-04      10         11
     2     7.93E-01     4.69E+01     1.40E-02     9.77E-04       0         12
     3     2.08E-01     2.04E+01     3.68E-03     9.77E-04       0         13
     4     8.38E-02     8.15E+00     1.48E-03     9.77E-04       0         14
     5     6.49E-02     3.39E+00     1.15E-03     9.77E-04       0         15
     6     6.15E-02     1.40E+00     1.09E-03     9.77E-04       0         16
     7     6.10E-02     5.98E-01     1.08E-03     9.77E-04       0         17
     8     6.08E-02     2.97E-01     1.08E-03     9.77E-04       0         18
     9     6.08E-02     2.07E-01     1.08E-03     9.77E-04       0         19
    10     6.07E-02     1.88E-01     1.08E-03     9.77E-04       0         20
```

Output file `iterate_ST.dat`.

4.2 Preconditioned nonlinear conjugate gradient: PNLCG

4.2.1 Variables declaration

The first step for using the PNLCG method is to declare the inputs and outputs of the function.

1. Include the header file optim.h to declare the data structure optim which will contain most of the information required by PNLCG.
2. Declare the integer n, dimension of the problem (2.1).
3. Declare the real `fcost`, the cost function f computed at x.
4. Declare the vector `x`, the unknown $x \in \mathcal{R}^n$ of the minimization problem (2.1).
5. Declare the vector `grad`, which corresponds to the gradient of the cost function f at x: $\nabla f(x) \in \mathcal{R}^n$.
6. Declare the vector `grad_preco`, which corresponds to the gradient of the cost function f at x multiplied by the preconditioner: $P_k \nabla f(x) \in \mathcal{R}^n$.
7. Declare the data structure optim.
8. Declare the chain of character of length 4 FLAG. This is the flag for the reverse communication between the PNLCG routine and the user.

```
implicit none
include 'optim_type.h'

integer :: n                                    ! dimension of the problem
real :: fcost                                   ! cost function value
real,dimension(:),allocatable :: x              ! current point
real,dimension(:),allocatable :: grad           ! gradient
real,dimension(:),allocatable :: grad_preco     ! preconditioned gradient
type(optim_type) :: optim                       ! data structure
character*4 :: FLAG                             ! communication FLAG
```

Declaration of the `FORTRAN` variables.

4.2.2 Initialization

The second step consists in initializing the problem.

1. Set the dimension n of the problem, n should be chosen such that $n \geq 1$.

2. Initialize the communication flag to 'INIT'.

3. Set the maximum number of nonlinear iteration that can be performed `optim%niter_max`.

4. Set the tolerance parameter for the stopping criterion `optim%conv`. The stopping criterion which is implemented by default is

$$(f(x)/f(x_0) < \texttt{optim\%conv}) \text{ OR } (\texttt{optim\%niter} \geq \texttt{optim\%niter\_max})\,. \tag{4.3}$$

   This means that the flag 'CONV' will be returned to the user when this condition is met. However, the user can define his own convergence criterion, as he defines himself the convergence loop (see next section).

5. Set the flag for printing output: `optim%print_flag` is set to 1 the output files containing information on the convergence history are created. No output files are created otherwise.

6. Set the level of of information in the output file `iterate_CG.dat`: if `optim%debug` is set to `true` then information concerning the linesearch process will be printed, otherwise, if it is set to `false`, the file `iterate_ST.dat` will contain only the convergence history (see 5.2.5 for more details).

7. Set the flag for using/not using bound constraints: if `optim%bound` is set to 1, bound constraints are used. If `optim%bound` is set to 0, no bound constraints are imposed.

8. If bound constraints have been activated, then the user must set additional variables. First, allocate the vectors `optim%ub` and `optim%lb` for respectively "upper bounds" and "lower bounds". The size of this vectors has to be equal to the dimension of the optimization problem n. Then set `optim%ub` and `optim%lb` with the corresponding bound constraints values. Each component i of these vectors define upper and lower bounds for the component x_i of the unknown. Finally, set the tolerance `optim%threshold`. This value gives the tolerance with which the bound constraints are satisfied. In practice, we enforce the condition

$$\texttt{optim\%lb}_i + \texttt{optim\%threshold} \leq x_i \leq \texttt{optim\%ub}_i - \texttt{optim\%threshold} \tag{4.4}$$

9. Define the initial guess: the unknown `x` has to be allocated and initialized to a specific value.

10. VERY IMPORTANT: compute the cost function and the gradient corresponding to the initial guess and store the result in `fcost` and `grad`

11. VERY IMPORTANT: multiply the gradient by the preconditioner and store the result in `grad_preco`. Note that if no preconditioner is available, you simply have to copy `grad` in `grad_preco`. This means that the preconditioner is identity. In this case the PNLCG method is a standard nonlinear conjugate gradient (without preconditioning).

It is very important that `fcost`, `grad` *and* `grad_preco` *are initialized to the values corresponding to the initial guess* `x` *on the first call to the solver PNLCG.*

```
n=2                              ! dimension
FLAG='INIT'                      ! first flag
optim%niter_max=10000            ! maximum iteration number
optim%conv=1e-8                  ! tolerance for the stopping criterion
optim%debug=.false.              ! level of details for output files
optim%pring_flag=1               ! print information
optim%bound=1                    ! activation of bound constraints
allocate(optim%ub(n))            ! allocate upper bound
allocate(optim%lb(n))            ! allocate lower bound
ub(:)=40.                        ! set upper bound
lb(:)=-40.                       ! set lower bound
optim%threshold=1e-0             ! set tolerance for bound constraints

allocate(x(n),grad(n),grad_preco(n)) ! allocation
x(1)=1.5                         ! set initial point
x(2)=1.5

call Rosenbrock(x,fcost,grad)    ! initial cost and gradient
grad_preco(:)=grad(:)            ! no preconditioning
```

Initialization.

4.2.3 Minimization within the reverse communication loop

The third step consists in performing the minimization within the reverse communication loop.

1. Define the `while` loop. In the chosen example, the loop is terminated when the convergence criterion is satisfied. In this case the communication flag is 'CONV'. The loop is also terminated when the flag returned by the PNLCG solver is 'FAIL', which indicates that the linesearch process has failed in finding a suitable steplength in the current descent direction.

2. At each iteration of the while loop, call the PNLCG routine. On first call, the flag is set to 'INIT', the unknown `x` is initialized to the initial guess x_0, and the variables `fcost`, `grad` and `grad_preco` contain respectively $f(x_0)$, $\nabla f(x_0)$ and $P_0 \nabla f(x_0)$.

3. On return of the call to the PNLCG routine, if the communication flag is 'GRAD' then the value of x has been modified. Compute the cost function $f(x)$ and the gradient $\nabla f(x)$ at this new point in the variables `fcost` and `grad`. The user also has the possibility of using his preconditioner. To do so he has to compute $P_k \nabla f(x)$ and store it in the variable `grad_preco`. Note that the preconditioner can change throughout the iterations. If no preconditioner is available, the user has to copy the gradient value $\nabla f(x)$ in `grad_preco`. In this case, the PNLCG method is equivalent to a nonlinear conjugate gradient method.

```
do while ((FLAG.ne.'CONV').and.(FLAG.ne.'FAIL'))
   call PNLCG(n,x,fcost,grad,grad_preco,optim,FLAG)
   if(FLAG.eq.'GRAD') then
      !compute cost and gradient at point x
      call Rosenbrock(x,fcost,grad)
      ! no preconditioning in this test: simply copy grad in
      ! grad_preco
      grad_preco(:)=grad(:)
   endif
enddo
```

Reverse communication loop.

4.2.4 End of the loop and output file

At the end of the reverse communication loop, either convergence has been reached and the communication flag is 'CONV', or the linesearch has failed and the communication flag is 'FAIL'. In both cases, the vector `x` contains the last value of the minimization sequence (2.3), the best approximation to the solution of the minimization problem that can be found using the PNLCG.

The convergence history is written in the file `iterate_PC.dat`. This file contains a remainder of the optimization settings: convergence criterion and maximum number of iteration. He also presents the initial cost without normalization and the initial norm of the gradient. Then, it presents on 7 columns the convergence history.

- Column 1 is for the nonlinear iteration number.
- Column 2 is for the non normalized cost function value.
- Column 3 is for the norm of the gradient.
- Column 4 is for the relative cost function value (normalized by the initial value, on the first iteration this value is then always equal to 1).
- Column 5 is for the size of the steplength taken.
- Column 6 is for the number of linesearch iteration for determining the steplength.
- Column 7 is for the total number of gradient computation.

```
*****************************************************************
          NONLINEAR CONJUGATE GRADIENT ALGORITHM
*****************************************************************
     Convergence criterion   :    1.00E-08
     Niter_max               :    10000
     Initial cost is         :    5.65E+01
     Initial norm_grad is    :    4.75E+02
*****************************************************************
   Niter      fk          ||gk||       fk/f0        alpha        nls      ngrad
     0     5.65E+01     4.75E+02     1.00E+00     1.00E+00       0         0
     1     2.74E+01     2.45E+02     4.86E-01     9.77E-04      10        11
     2     4.65E+00     1.21E+02     8.22E-02     4.88E-04       1        13
     3     9.28E-02     9.19E+00     1.64E-03     4.88E-04       0        14
     4     6.54E-02     2.07E+00     1.16E-03     4.88E-04       0        15
     5     6.40E-02     2.49E-01     1.13E-03     4.88E-04       0        16
     6     6.39E-02     1.89E-01     1.13E-03     4.88E-04       0        17
     7     6.39E-02     2.56E-01     1.13E-03     4.88E-04       0        18
     8     6.39E-02     3.41E-01     1.13E-03     4.88E-04       0        19
     9     6.36E-02     8.47E-01     1.13E-03     4.88E-04       0        20
    10     6.31E-02     1.19E+00     1.12E-03     4.88E-04       0        21
```

Output file `iterate_CG.dat`.

4.3 Quasi-Newton l-BFGS method: LBFGS

4.3.1 Variables declaration

The first step for using the LBFGS method is to declare the inputs and outputs of the function.

1. Include the header file optim.h to declare the data structure optim which will contain most of the information required by LBFGS.

2. Declare the integer n, dimension of the problem (2.1).

3. Declare the real `fcost`, the cost function f computed at x.

4. Declare the vector `x`, the unknown $x \in \mathcal{R}^n$ of the minimization problem (2.1).

5. Declare the vector `grad`, which corresponds to the gradient of the cost function f at x: $\nabla f(x) \in \mathcal{R}^n$.

6. Declare the data structure optim.

7. Declare the chain of character of length 4 FLAG. This is the flag for the reverse communication between the LBFGS routine and the user.

```
implicit none
include 'optim_type.h'

integer :: n                                    ! dimension of the problem
real :: fcost                                   ! cost function value
real,dimension(:),allocatable :: x              ! current point
real,dimension(:),allocatable :: grad           ! current gradient
type(optim_type) :: optim                       ! data structure
character*4 :: FLAG                             ! communication FLAG
```

Declaration of the `FORTRAN` variables.

4.3.2 Initialization

The second step consists in initializing the problem.

1. Set the dimension n of the problem, n should be chosen such that $n \geq 1$.

2. Initialize the communication flag to 'INIT'.

3. Set the maximum number of nonlinear iteration that can be performed `optim%niter_max`.

4. Set the tolerance parameter for the stopping criterion `optim%conv`. The stopping criterion which is implemented by default is

$$(f(x)/f(x_0) < \texttt{optim\%conv}) \text{ OR } (\texttt{optim\%niter} \geq \texttt{optim\%niter\_max})\,. \tag{4.5}$$

   This means that the flag 'CONV' will be returned to the user when this condition is met. However, the user can define his own convergence criterion, as he defines himself the convergence loop (see next section).

5. Set the flag for printing output: `optim%print_flag` is set to 1 the output files containing information on the convergence history are created. No output files are created otherwise.

6. Set the level of of information in the output file `iterate_LB.dat`: if `optim%debug` is set to `true` then information concerning the linesearch process will be printed, otherwise, if it is set to `false`, the file `iterate_LB.dat` will contain only the convergence history (see 5.2.5 for more details).

7. Set the maximum number of pairs of vectors which will be used to compute the l-BFGS approximation of the inverse Hessian. This is the variable `optim%l`. Usual choices for l go from 3 to 40.

8. Set the flag for using/not using bound constraints: if `optim%bound` is set to 1, bound constraints are used. If `optim%bound` is set to 0, no bound constraints are imposed.

9. If bound constraints have been activated, then the user must set additional variables. First, allocate the vectors `optim%ub` and `optim%lb` for respectively "upper bounds" and "lower bounds". The size of this vectors has to be equal to the dimension of the optimization problem n. Then set `optim%ub` and `optim%lb` with the corresponding bound constraints values. Each component i of these vectors define upper and lower bounds for the component x_i of the unknown. Finally, set the tolerance `optim%threshold`. This value gives the tolerance with which the bound constraints are satisfied. In practice, we enforce the condition

$$\texttt{optim\%lb}_i + \texttt{optim\%threshold} \leq x_i \leq \texttt{optim\%ub}_i - \texttt{optim\%threshold} \tag{4.6}$$

10. Define the initial guess: the unknown `x` has to be allocated and initialized to a specific value.

11. VERY IMPORTANT: compute the cost function and the gradient corresponding to the initial guess and store the result in `fcost` and `grad`

It is very important that `fcost`*, and* `grad` *are initialized to the values corresponding to the initial guess* `x` *on the first call to the solver LBFGS.*

```
n=2                          ! dimension
FLAG='INIT'                  ! first flag
optim%niter_max=10000        ! maximum iteration number
optim%conv=1e-8              ! tolerance for the stopping criterion
optim%debug=.false.          ! level of details for output files
optim%l=20                   ! maximum number of stored pairs used for
                             ! the l-BFGS approximation
optim%pring_flag=1           ! print information
optim%bound=1                ! activation of bound constraints
allocate(optim%ub(n))        ! allocate upper bound
allocate(optim%lb(n))        ! allocate lower bound
ub(:)=40.                    ! set upper bound
lb(:)=-40.                   ! set lower bound
optim%threshold=1e-2         ! set tolerance for bound constraints

allocate(x(n),grad(n),grad_preco(n)) ! allocation
x(1)=1.5                             ! set initial point
x(2)=1.5

call Rosenbrock(x,fcost,grad)        ! initial cost and gradient
```

Initialization.

4.3.3 Minimization within the reverse communication loop

The third step consists in performing the minimization within the reverse communication loop.

1. Define the `while` loop. In the chosen example, the loop is terminated when the convergence criterion is satisfied. In this case the communication flag is 'CONV'. The loop is also terminated when the flag returned by the LBFGS solver is 'FAIL', which indicates that the linesearch process has failed in finding a suitable steplength in the current descent direction.

2. At each iteration of the while loop, call the LBFGS routine. On first call, the flag is set to 'INIT', the unknown `x` is initialized to the initial guess x_0, and the variables `fcost` and `grad` contain respectively $f(x_0)$ and $\nabla f(x_0)$.

3. On return of the call to the LBFGS routine, if the communication flag is 'GRAD' then the value of x has been modified. Compute the cost function $f(x)$ and the gradient $\nabla f(x)$ at this new point in the variables `fcost` and `grad`.

```
do while ((FLAG.ne.'CONV').and.(FLAG.ne.'FAIL'))
   call LBFGS(n,x,fcost,grad,optim,FLAG)
   if(FLAG.eq.'GRAD') then
      !compute cost and gradient at point x
      call Rosenbrock(x,fcost,grad)
   endif
enddo
```

```
*****************************************************************
                    l-BFGS ALGORITHM
*****************************************************************
     Convergence criterion   :    1.00E-08
     Niter_max               :    10000
     Initial cost is         :    5.65E+01
     Initial norm_grad is    :    4.75E+02
*****************************************************************
   Niter       fk          ||gk||       fk/f0        alpha        nls      ngrad
     0     5.65E+01     4.75E+02     1.00E+00     1.00E+00        0          0
     1     2.74E+01     2.45E+02     4.86E-01     9.77E-04       10         11
     2     2.12E+00     7.47E+01     3.75E-02     9.77E-01        3         15
     3     1.67E-01     1.75E+01     2.96E-03     9.77E-01        0         16
     4     6.37E-02     5.76E-01     1.13E-03     9.77E-01        0         17
     5     6.36E-02     1.89E-01     1.12E-03     9.77E-01        0         18
     6     6.35E-02     1.94E-01     1.12E-03     9.77E-01        0         19
     7     6.33E-02     5.40E-01     1.12E-03     9.77E-01        0         20
     8     6.28E-02     1.06E+00     1.11E-03     9.77E-01        0         21
     9     6.15E-02     2.00E+00     1.09E-03     9.77E-01        0         22
    10     5.86E-02     3.31E+00     1.04E-03     9.77E-01        0         23
```

Output file `iterate_LB.dat`.

Reverse communication loop.

4.3.4 End of the loop and output file

At the end of the reverse communication loop, either convergence has been reached and the communication flag is 'CONV', or the linesearch has failed and the communication flag is 'FAIL'. In both cases, the vector `x` contains the last value of the minimization sequence (2.3), the best approximation to the solution of the minimization problem that can be found using LBFGS.

The convergence history is written in the file `iterate_LB.dat`. This file contains a remainder of the optimization settings: convergence criterion and maximum number of iteration. He also presents the initial cost without normalization and the initial norm of the gradient. Then, it presents on 7 columns the convergence history.

- Column 1 is for the nonlinear iteration number.
- Column 2 is for the non normalized cost function value.
- Column 3 is for the norm of the gradient.
- Column 4 is for the relative cost function value (normalized by the initial value, on the first iteration this value is then always equal to 1).
- Column 5 is for the size of the steplength taken.
- Column 6 is for the number of linesearch iteration for determining the steplength.
- Column 7 is for the total number of gradient computation.

4.4 Quasi-Newton preconditioned l-BFGS method: PLBFGS

4.4.1 Variables declaration

The first step for using the PLBFGS method is to declare the inputs and outputs of the function.

1. Include the header file optim.h to declare the data structure optim which will contain most of the information required by PLBFGS.
2. Declare the integer n, dimension of the problem (2.1).
3. Declare the real `fcost`, the cost function f computed at x.
4. Declare the vector `x`, the unknown $x \in \mathcal{R}^n$ of the minimization problem (2.1).
5. Declare the vector `grad`, which corresponds to the gradient of the cost function f at x: $\nabla f(x) \in \mathcal{R}^n$.
6. Declare the vector `grad_preco`, which corresponds to the gradient of the cost function f at x multiplied by the preconditioner: $P_k \nabla f(x) \in \mathcal{R}^n$. This structure is used only at the first iteration.
7. Declare the data structure optim.
8. Declare the chain of character of length 4 FLAG. This is the flag for the reverse communication between the PLBFGS routine and the user.

```
implicit none
include 'optim_type.h'

integer :: n                                    ! dimension of the problem
real :: fcost                                   ! cost function value
real,dimension(:),allocatable :: x              ! current point
real,dimension(:),allocatable :: grad           ! current gradient
type(optim_type) :: optim                       ! data structure for the optimizer
character*4 :: FLAG                             ! communication FLAG
```

Declaration of the `FORTRAN` variables.

4.4.2 Initialization

The second step consists in initializing the problem.

1. Set the dimension n of the problem, n should be chosen such that $n \geq 1$.
2. Initialize the communication flag to 'INIT'.
3. Set the maximum number of nonlinear iteration that can be performed `optim%niter_max`.

4. Set the tolerance parameter for the stopping criterion `optim%conv`. The stopping criterion which is implemented by default is

$$(f(x)/f(x_0) < \texttt{optim\%conv}) \text{ OR } (\texttt{optim\%niter} \geq \texttt{optim\%niter\_max}) . \tag{4.7}$$

   This means that the flag 'CONV' will be returned to the user when this condition is met. However, the user can define his own convergence criterion, as he defines himself the convergence loop (see next section).

5. Set the flag for printing output: `optim%print_flag` is set to 1 the output files containing information on the convergence history are created. No output files are created otherwise.

6. Set the level of of information in the output file `iterate_PLB.dat`: if `optim%debug` is set to `true` then information concerning the linesearch process will be printed, otherwise, if it is set to `false`, the file `iterate_LB.dat` will contain only the convergence history (see 5.2.5 for more details).

7. Set the maximum number of pairs of vectors which will be used to compute the l-BFGS approximation of the inverse Hessian. This is the variable `optim%l`. Usual choices for l go from 3 to 40. item Set the flag for using/not using bound constraints: if `optim%bound` is set to 1, bound constraints are used. If `optim%bound` is set to 0, no bound constraints are imposed.

8. If bound constraints have been activated, then the user must set additional variables. First, allocate the vectors `optim%ub` and `optim%lb` for respectively "upper bounds" and "lower bounds". The size of this vectors has to be equal to the dimension of the optimization problem n. Then set `optim%ub` and `optim%lb` with the corresponding bound constraints values. Each component i of these vectors define upper and lower bounds for the component x_i of the unknown. Finally, set the tolerance `optim%threshold`. This value gives the tolerance with which the bound constraints are satisfied. In practice, we enforce the condition

$$\texttt{optim\%lb}_i + \texttt{optim\%threshold} \leq x_i \leq \texttt{optim\%ub}_i - \texttt{optim\%threshold} \tag{4.8}$$

9. Define the initial guess: the unknown `x` has to be allocated and initialized to a specific value.

10. VERY IMPORTANT: compute the cost function and the gradient corresponding to the initial guess and store the result in `fcost` and `grad`

11. VERY IMPORTANT: apply the preconditioner ONLY AT FIRST ITERATION on the gradient; `grad_preco` should contain $P_0 \nabla f(x_0)$ at initialization. This has to be done only at initialization. The preconditioning operations within the loop need not act on the gradient `grad`.

It is very important that `fcost`, `grad` and `grad_preco` are initialized to the values corresponding to the initial guess `x` on the first call to the solver PLBFGS.

```
n=2                              ! dimension
FLAG='INIT'                      ! first flag
optim%niter_max=10000            ! maximum iteration number
optim%conv=1e-8                  ! tolerance for the stopping criterion
optim%debug=.false.              ! level of details for output files
optim%l=20                       ! maximum number of stored pairs used for
                                 ! the l-BFGS approximation
optim%pring_flag=1               ! print information
optim%bound=1                    ! activation of bound constraints
allocate(optim%ub(n))            ! allocate upper bound
allocate(optim%lb(n))            ! allocate lower bound
ub(:)=40.                        ! set upper bound
lb(:)=-40.                       ! set lower bound
optim%threshold=1e-2             ! set tolerance for bound constraints

allocate(x(n),grad(n),grad_preco(n)) ! allocation
x(1)=1.5                         ! set initial point
x(2)=1.5

call Rosenbrock(x,fcost,grad)    ! initial cost and gradient
grad_preco(:)=grad(:)            ! no preconditioning
```

Initialization.

4.4.3 Minimization within the reverse communication loop

The third step consists in performing the minimization within the reverse communication loop.

1. Define the `while` loop. In the chosen example, the loop is terminated when the convergence criterion is satisfied. In this case the communication flag is 'CONV'. The loop is also terminated when the flag returned by the PLBFGS solver is 'FAIL', which indicates that the linesearch process has failed in finding a suitable steplength in the current descent direction.

2. At each iteration of the while loop, call the PLBFGS routine. On first call, the flag is set to 'INIT', the unknown `x` is initialized to the initial guess x_0, and the variables `fcost`, `grad` and `grad_preco` contain respectively $f(x_0)$, $\nabla f(x_0)$ and $P_0 \nabla f(x_0)$.

3. On return of the call to the PLBFGS routine, if the communication flag is 'GRAD' then the value of x has been modified. Compute the cost function $f(x)$ and the gradient $\nabla f(x)$ at this new point in the variables `fcost` and `grad`.

4. On return of the call to the PLBFGS routine, if the communication flag is 'PREC' then the user has the possibility of applying his preconditioner. To do so, the user must multiply the vector `optim%q_plb` by the preconditioner P_k, and store the result directly in `optim%q_plb`. The old value of `optim%q_plb` does not have to be stored. If nothing is done at this stage, the PLBFGS method is equivalent to the LBFGS method (no preconditioning is applied). Note that the preconditioner can change throughout the iterations.

```
do while ((FLAG.ne.'CONV').and.(FLAG.ne.'FAIL'))
   call PLBFGS(n,x,fcost,grad,grad_preco,optim,FLAG)
   if(FLAG.eq.'GRAD') then
      call Rosenbrock(x,fcost,grad)
   endif
   if(FLAG.eq.'PREC') then
      !apply preconditioning to optim%q_plb
      !if nothing is done, PLBFGS is equivalent to LBFGS
      optim%q_plb(:)=optim%q_plb(:)
   endif
enddo
```

Reverse communication loop.

4.4.4 End of the loop and output file

At the end of the reverse communication loop, either convergence has been reached and the communication flag is 'CONV', or the linesearch has failed and the communication flag is 'FAIL'. In both cases, the vector `x` contains the last value of the minimization sequence (2.3), the best approximation to the solution of the minimization problem that can be found using PLBFGS.

The convergence history is written in the file `iterate_PLB.dat`. This file contains a remainder of the optimization settings: convergence criterion and maximum number of iteration. He also presents the initial cost without normalization and the initial norm of the gradient. Then, it presents on 7 columns the convergence history.

- Column 1 is for the nonlinear iteration number.
- Column 2 is for the non normalized cost function value.
- Column 3 is for the norm of the gradient.
- Column 4 is for the relative cost function value (normalized by the initial value, on the first iteration this value is then always equal to 1).
- Column 5 is for the size of the steplength taken.
- Column 6 is for the number of linesearch iteration for determining the steplength.
- Column 7 is for the total number of gradient computation.

```
*****************************************************************
          PRECONDITIONED l-BFGS ALGORITHM
*****************************************************************
     Convergence criterion   :   1.00E-08
     Niter_max               :   10000
     Initial cost is         :   5.65E+01
     Initial norm_grad is    :   4.75E+02
*****************************************************************
   Niter      fk          ||gk||       fk/f0       alpha       nls     ngrad
     0    5.65E+01    4.75E+02    1.00E+00    1.00E+00      0        0
     1    2.74E+01    2.45E+02    4.86E-01    9.77E-04     10       11
     2    2.12E+00    7.47E+01    3.75E-02    9.77E-01      3       15
     3    1.67E-01    1.75E+01    2.96E-03    9.77E-01      0       16
     4    6.37E-02    5.76E-01    1.13E-03    9.77E-01      0       17
     5    6.36E-02    1.89E-01    1.12E-03    9.77E-01      0       18
     6    6.35E-02    1.94E-01    1.12E-03    9.77E-01      0       19
     7    6.33E-02    5.40E-01    1.12E-03    9.77E-01      0       20
     8    6.28E-02    1.06E+00    1.11E-03    9.77E-01      0       21
     9    6.15E-02    2.00E+00    1.09E-03    9.77E-01      0       22
    10    5.86E-02    3.31E+00    1.04E-03    9.77E-01      0       23
```

Output file `iterate_PLB.dat`.

4.5 Truncated Newton method: TRN

4.5.1 Variables declaration

The first step for using the TRN method is to declare the inputs and outputs of the function.

1. Include the header file optim.h to declare the data structure optim which will contain most of the information required by TRN.

2. Declare the integer n, dimension of the problem (2.1).

3. Declare the real `fcost`, the cost function f computed at x.

4. Declare the vector `x`, the unknown $x \in \mathcal{R}^n$ of the minimization problem (2.1).

5. Declare the vector `grad`, which corresponds to the gradient of the cost function f at x: $\nabla f(x) \in \mathcal{R}^n$.

6. Declare the data structure optim.

7. Declare the chain of character of length 4 FLAG. This is the flag for the reverse communication between the TRN routine and the user.

```
implicit none
include 'optim_type.h'

integer :: n                                    ! dimension of the problem
real :: fcost                                   ! cost function value
real,dimension(:),allocatable :: x              ! current point
real,dimension(:),allocatable :: grad           ! current gradient
type(optim_type) :: optim                       ! data structure for the optimizer
character*4 :: FLAG                             ! communication FLAG
```

Declaration of the `FORTRAN` variables.

4.5.2 Initialization

The second step consists in initializing the problem.

1. Set the dimension n of the problem, n should be chosen such that $n \geq 1$.

2. Initialize the communication flag to 'INIT'.

3. Set the maximum number of nonlinear iteration that can be performed `optim%niter_max`.

4. Set the tolerance parameter for the stopping criterion `optim%conv`. The stopping criterion which is implemented by default is

$$(f(x)/f(x_0) < \texttt{optim\%conv}) \text{ OR } (\texttt{optim\%niter} \geq \texttt{optim\%niter\_max}) . \tag{4.9}$$

   This means that the flag 'CONV' will be returned to the user when this condition is met. However, the user can define his own convergence criterion, as he defines himself the convergence loop (see next section).

5. Set the flag for printing output: `optim%print_flag` is set to 1 the output files containing information on the convergence history are created. No output files are created otherwise.

6. Set the level of of information in the output files `iterate_TRN.dat` and `iterate_TRN_CG.dat`: if `optim%debug` is set to `true` then information concerning the linesearch process will be printed in `iterate_TRN.dat`. In addition, extra information on the convergence of the inner iterations through the conjugate gradient algorithm will be printed in `iterate_TRN_CG.dat` (namely the decrease of the quadratic form associated to the symmetric definite linear system). Otherwise, if it is set to `false`, the files `iterate_TRN.dat` and `iterate_TRN_CG.dat` will contain only the convergence history (see 5.2.5 and 5.4.2 for more details).

7. Set the maximum number of iterations `optim%niter_CG_max` for the resolution of the inner linear system using the matrix-free conjugate gradient algorithm. This linear system is solved to compute an approximation of the Newton descent direction. item Set the flag for using/not using bound constraints: if `optim%bound` is set to 1, bound constraints are used. If `optim%bound` is set to 0, no bound constraints are imposed.

8. If bound constraints have been activated, then the user must set additional variables. First, allocate the vectors `optim%ub` and `optim%lb` for respectively "upper bounds" and "lower bounds". The size of this vectors has to be equal to the dimension of the optimization problem n. Then set `optim%ub` and `optim%lb` with the corresponding bound constraints values. Each component i of these vectors define upper and lower bounds for the component x_i of the unknown. Finally, set the tolerance `optim%threshold`. This value gives the tolerance with which the bound constraints are satisfied. In practice, we enforce the condition

$$\texttt{optim\%lb}_i + \texttt{optim\%threshold} \leq x_i \leq \texttt{optim\%ub}_i - \texttt{optim\%threshold} \tag{4.10}$$

9. Define the initial guess: the unknown `x` has to be allocated and initialized to a specific value.

10. VERY IMPORTANT: compute the cost function and the gradient corresponding to the initial guess and store the result in `fcost` and `grad`

It is very important that ***fcost****, and* ***grad*** *are initialized to the values corresponding to the initial guess* ***x*** *on the first call to the solver TRN.*

```
n=2                              ! dimension
FLAG='INIT'                      ! first flag
optim%niter_max=100              ! maximum iteration number
optim%conv=1e-8                  ! tolerance for the stopping criterion
optim%debug=.false.              ! level of details for output files
optim%niter_max_CG=5             ! maximum number of inner conjugate gradient
                                 ! iterations
optim%pring_flag=1               ! print information
optim%bound=1                    ! activation of bound constraints
allocate(optim%ub(n))            ! allocate upper bound
allocate(optim%lb(n))            ! allocate lower bound
ub(:)=40.                        ! set upper bound
lb(:)=-40.                       ! set lower bound
optim%threshold=1e-2             ! set tolerance for bound constraints

allocate(x(n),grad(n),grad_preco(n)) ! allocation
x(1)=1.5                             ! set initial point
x(2)=1.5

call Rosenbrock(x,fcost,grad)        ! initial cost and gradient
```

Initialization.

4.5.3 Minimization within the reverse communication loop

The third step consists in performing the minimization within the reverse communication loop.

1. Define the `while` loop. In the chosen example, the loop is terminated when the convergence criterion is satisfied. In this case the communication flag is 'CONV'. The loop is also terminated when the flag returned by the TRN solver is 'FAIL', which indicates that the linesearch process has failed in finding a suitable steplength in the current descent direction.

2. At each iteration of the while loop, call the TRN routine. On first call, the flag is set to 'INIT', the unknown `x` is initialized to the initial guess x_0, and the variables `fcost` and `grad` contain respectively $f(x_0)$ and $\nabla f(x_0)$.

3. On return of the call to the TRN routine, if the communication flag is 'GRAD' then the value of x has been modified. Compute the cost function $f(x)$ and the gradient $\nabla f(x)$ at this new point in the variables `fcost` and `grad`.

4. On return of the call to the TRN routine, if the communication flag is 'HESS' then the user is requested for performing one Hessian-vector product for the resolution of the inner linear system. The vector to multiply is in `optim%d`. The result of the multiplication of

this vector by the Hessian operator has to be stored in the varialbe `optim%Hd`. Be careful not to modify the value of the vector `optim%d`.

```
do while ((FLAG.ne.'CONV').and.(FLAG.ne.'FAIL'))
   call TRN(n,x,fcost,grad,optim,FLAG)
   if(FLAG.eq.'GRAD') then
      call Rosenbrock(x,fcost,grad)
   elseif(FLAG.eq.'HESS') then
      call Rosenbrock_Hess(x,optim%d,optim%Hd)
   endif
enddo
```

Reverse communication loop.

4.5.4 End of the loop and output file

At the end of the reverse communication loop, either convergence has been reached and the communication flag is 'CONV', or the linesearch has failed and the communication flag is 'FAIL'. In both cases, the vector `x` contains the last value of the minimization sequence (2.3), the best approximation to the solution of the minimization problem that can be found using TRN.

The convergence history is written in the file `iterate_TRN.dat`. This file contains a remainder of the optimization settings: convergence criterion and maximum number of iterations. He also presents the initial cost without normalization and the initial norm of the gradient. Then, it presents on 10 columns the convergence history.

- Column 1 is for the nonlinear iteration number.
- Column 2 is for the non normalized cost function value.
- Column 3 is for the norm of the gradient.
- Column 4 is for the relative cost function value (normalized by the initial value, on the first iteration this value is then always equal to 1).
- Column 5 is for the size of the steplength taken.
- Column 6 is for the number of linesearch iteration for determining the steplength.
- Column 7 is for the number of conjugate gradient iteration used to compute the descent direction
- Column 8 is for the forcing term η used to define the stopping criterion for the inner iterations.
- Column 9 is for the total number of gradient computation.
- Column 10 is for the total number of Hessian-vector products.

```
*********************************************************************************************
                                TRUNCATED NEWTON ALGORITHM
*********************************************************************************************
     Convergence criterion   :    1.00E-08
     Niter_max               :      100
     Initial cost is         :    5.65E+01
     Initial norm_grad is    :    4.75E+02
     Maximum CG iter         :       5
*********************************************************************************************
   Niter      fk          ||gk||       fk/f0        alpha        nls   nit_CG     eta        ngrad     nhess
     0     5.65E+01    4.75E+02    1.00E+00    1.00E+00        0        0     9.00E-01         0         0
     1     1.73E+00    7.19E+01    3.05E-02    1.00E+00        0        1     9.00E-01         2         1
     2     6.94E-02    2.92E+00    1.23E-03    1.00E+00        0        1     8.43E-01         3         2
     3     6.65E-02    1.90E-01    1.18E-03    1.00E+00        0        1     7.59E-01         4         3
     4     3.51E-02    3.14E+00    6.21E-04    2.50E-01        2        2     6.40E-01         7         5
     5     3.35E-02    2.36E+00    5.92E-04    2.50E-01        0        1     9.00E-01         8         6
     6     3.25E-02    1.77E+00    5.76E-04    2.50E-01        0        1     8.43E-01         9         7
     7     3.20E-02    1.33E+00    5.67E-04    2.50E-01        0        1     7.59E-01        10         8
     8     3.17E-02    9.97E-01    5.61E-04    2.50E-01        0        1     7.52E-01        11         9
     9     3.15E-02    7.49E-01    5.58E-04    2.50E-01        0        1     7.54E-01        12        10
    10     3.14E-02    5.62E-01    5.57E-04    2.50E-01        0        1     7.57E-01        13        11
```

Output file `iterate_TRN.dat`.

Additional information on the convergence of the inner linear system is written in the file `iterate_TRN_CG.dat`. This file contains a remainder of the optimization settings: convergence criterion, maximum number of iterations, maximum number of iteration for the resolution of the inner linear systems. He also presents the initial cost without normalization and the initial norm of the gradient. Then, for each nonlinear iteration, a convergence history of the inner liner system using the matrix-free conjugate gradient is presented. The number of the nonlinear iteration appears at the top, as well as the value of the forcing term η for this nonlinear iteration. Then, the convergence history is presented on 4 columns.

- Column 1 is for the iteration number
- Column 2 is for the value of the quadratic function associated with the linear system, which is supposed to be symmetric definite. This information is available only if the option `optim%debug` is set to `true`. If it is set to `false`, only 0 is written as a default value.
- Column 3 is for the norm of the residuals of the inner system.
- Column 4 is for the relative residuals value. When this value becomes lower than η, the stopping criterion is satisfied.

```
*******************************************************************************
                          TRUNCATED NEWTON ALGORITHM
                             INNER CG HISTORY
*******************************************************************************
     Convergence criterion   :    1.00E-08
     Niter_max               :      100
     Initial cost is         :    5.65E+01
     Initial norm_grad is    :    4.75E+02
     Maximum CG iter         :       5
*******************************************************************************
-------------------------------------------------------------------------------
 NONLINEAR ITERATION    0 ETA IS :      9.00E-01
-------------------------------------------------------------------------------
  Iter_CG      qk         norm_res     norm_res/||gk||
       0     0.00E+00     4.75E+02      1.00E+00
       1     0.00E+00     1.86E+01      3.92E-02
-------------------------------------------------------------------------------
 NONLINEAR ITERATION    1 ETA IS :      8.43E-01
-------------------------------------------------------------------------------
  Iter_CG      qk         norm_res     norm_res/||gk||
       0     0.00E+00     7.19E+01      1.00E+00
       1     0.00E+00     5.83E-01      8.11E-03
-------------------------------------------------------------------------------
 NONLINEAR ITERATION    2 ETA IS :      7.59E-01
-------------------------------------------------------------------------------
  Iter_CG      qk         norm_res     norm_res/||gk||
       0     0.00E+00     2.92E+00      1.00E+00
       1     0.00E+00     1.89E-01      6.49E-02
-------------------------------------------------------------------------------
 NONLINEAR ITERATION    3 ETA IS :      6.40E-01
-------------------------------------------------------------------------------
  Iter_CG      qk         norm_res     norm_res/||gk||
       0     0.00E+00     1.90E-01      1.00E+00
       1     0.00E+00     4.82E+00      2.53E+01
       2     0.00E+00     1.63E-04      8.55E-04
```

Output file `iterate_TRN_CG.dat`.

4.6 Preconditioned truncated Newton method: PTRN

4.6.1 Variables declaration

The first step for using the PTRN method is to declare the inputs and outputs of the function.

1. Include the header file optim.h to declare the data structure optim which will contain most of the information required by TRN.

2. Declare the integer n, dimension of the problem (2.1).

3. Declare the real `fcost`, the cost function f computed at x.

4. Declare the vector `x`, the unknown $x \in \mathcal{R}^n$ of the minimization problem (2.1).

5. Declare the vector `grad`, which corresponds to the gradient of the cost function f at x: $\nabla f(x) \in \mathcal{R}^n$.

6. Declare the vector `grad_preco`, which corresponds to the gradient of the cost function f at x multiplied by the preconditioner: $P_k \nabla f(x) \in \mathcal{R}^n$.

7. Declare the data structure optim.

8. Declare the chain of character of length 4 FLAG. This is the flag for the reverse communication between the TRN routine and the user.

```
implicit none
include 'optim_type.h'

integer :: n                                    ! dimension of the problem
real :: fcost                                   ! cost function value
real,dimension(:),allocatable :: x              ! current point
real,dimension(:),allocatable :: grad           ! current gradient
real,dimension(:),allocatable :: grad_preco     ! preconditioned current gradient
type(optim_type) :: optim                       ! data structure for the optimizer
character*4 :: FLAG                             ! communication FLAG
```

Declaration of the FORTRAN variables.

4.6.2 Initialization

The second step consists in initializing the problem.

1. Set the dimension n of the problem, n should be chosen such that $n \geq 1$.

2. Initialize the communication flag to 'INIT'.

3. Set the maximum number of nonlinear iteration that can be performed `optim%niter_max`.

4. Set the tolerance parameter for the stopping criterion `optim%conv`. The stopping criterion which is implemented by default is

$$(f(x)/f(x_0) < \texttt{optim\%conv}) \text{ OR } (\texttt{optim\%niter} \geq \texttt{optim\%niter\_max}). \qquad (4.11)$$

   This means that the flag 'CONV' will be returned to the user when this condition is met. However, the user can define his own convergence criterion, as he defines himself the convergence loop (see next section).

5. Set the flag for printing output: `optim%print_flag` is set to 1 the output files containing information on the convergence history are created. No output files are created otherwise.

6. Set the level of of information in the output files `iterate_PTRN.dat` and `iterate_PTRN_CG.dat`: if `optim%debug` is set to `true` then information concerning the linesearch process will be printed in `iterate_PTRN.dat`. In addition, extra information on the convergence of the inner iterations through the conjugate gradient algorithm will be printed in `iterate_PTRN_CG.dat` (namely the decrease of the quadratic form associated to the symmetric definite linear system). Otherwise, if it is set to `false`, the files `iterate_PTRN.dat` and `iterate_PTRN_CG.dat` will contain only the convergence history (see 5.2.5 and 5.4.2 for more details).

7. Set the maximum number of iterations `optim%niter_CG_max` for the resolution of the inner linear system using the matrix-free conjugate gradient algorithm. This linear system is solved to compute an approximation of the Newton descent direction. item Set the flag for using/not using bound constraints: if `optim%bound` is set to 1, bound constraints are used. If `optim%bound` is set to 0, no bound constraints are imposed.

8. If bound constraints have been activated, then the user must set additional variables. First, allocate the vectors `optim%ub` and `optim%lb` for respectively "upper bounds" and "lower bounds". The size of this vectors has to be equal to the dimension of the optimization problem n. Then set `optim%ub` and `optim%lb` with the corresponding bound constraints values. Each component i of these vectors define upper and lower bounds for the component x_i of the unknown. Finally, set the tolerance `optim%threshold`. This value gives the tolerance with which the bound constraints are satisfied. In practice, we enforce the condition

$$\texttt{optim\%lb}_i + \texttt{optim\%threshold} \leq x_i \leq \texttt{optim\%ub}_i - \texttt{optim\%threshold} \tag{4.12}$$

9. Define the initial guess: the unknown `x` has to be allocated and initialized to a specific value.

10. VERY IMPORTANT: compute the cost function and the gradient corresponding to the initial guess and store the result in `fcost` and `grad`

11. VERY IMPORTANT: multiply the gradient by the preconditioner and store the result in `grad_preco`. Note that if no preconditioner is available, you simply have to copy `grad` in `grad_preco`. This means that the preconditioner is identity. In this case the PTRN method is a standard TNR method (without preconditioning).

It is very important that `fcost`, `grad` and `grad_preco` are initialized to the values corresponding to the initial guess `x` on the first call to the solver PTRN.

```
n=2                          ! dimension
FLAG='INIT'                  ! first flag
optim%niter_max=100          ! maximum iteration number
optim%conv=1e-8              ! tolerance for the stopping criterion
optim%debug=.false.          ! level of details for output files
optim%niter_max_CG=5         ! maximum number of inner conjugate gradient
                             ! iterations
optim%pring_flag=1           ! print information
optim%bound=1                ! activation of bound constraints
allocate(optim%ub(n))        ! allocate upper bound
allocate(optim%lb(n))        ! allocate lower bound
ub(:)=40.                    ! set upper bound
lb(:)=-40.                   ! set lower bound
optim%threshold=1e-2         ! set tolerance for bound constraints

allocate(x(n),grad(n),grad_preco(n)) ! allocation
x(1)=1.5                             ! set initial point
x(2)=1.5

call Rosenbrock(x,fcost,grad)        ! initial cost and gradient
grad_preco(:)=grad(:)                ! no preconditioning
```

Initialization.

4.6.3 Minimization within the reverse communication loop

The third step consists in performing the minimization within the reverse communication loop.

1. Define the `while` loop. In the chosen example, the loop is terminated when the convergence criterion is satisfied. In this case the communication flag is 'CONV'. The loop is also terminated when the flag returned by the PTRN solver is 'FAIL', which indicates that the linesearch process has failed in finding a suitable steplength in the current descent direction.

2. At each iteration of the while loop, call the PTRN routine. On first call, the flag is set to 'INIT', the unknown `x` is initialized to the initial guess x_0, and the variables `fcost`, `grad` and `grad_preco` contain respectively $f(x_0)$, $\nabla f(x_0)$ and $P_0 \nabla f(x_0)$.

3. On return of the call to the PTRN routine, if the communication flag is 'GRAD' then the value of x has been modified. Compute the cost function $f(x)$ and the gradient $\nabla f(x)$ at this new point in the variables `fcost` and `grad`. The user also has the possibility of using his preconditioner. To do so he has to compute $P_k \nabla f(x)$ and store it in the variable `grad_preco`. Note that the preconditioner can change throughout the iterations. If no preconditioner is available, the user has to copy the gradient value $\nabla f(x)$ in `grad_preco`.

4. On return of the call to the PTRN routine, if the communication flag is 'HESS' then the user is requested for performing one Hessian-vector product for the resolution of the inner linear system. The vector to multiply is in `optim%d`. The result of the multiplication of this vector by the Hessian operator has to be stored in the variable `optim%Hd`. Be careful not to modify the value of the vector `optim%d`.

5. On return of the call to the PTRN routine, if the communication flag is 'PREC' then the user has the possibility of applying his preconditioner. To do so, the user must multiply the vector `optim%residual` by the preconditioner, and store the result in `optim%residual_preco`. Be careful not to modify the value of the variable `optim%residual`. If no preconditioner is available, the use has to copy the value of `optim%residual` into `optim%residual_preco`. Note that the preconditioner can change throughout the iterations.

```
do while ((FLAG.ne.'CONV').and.(FLAG.ne.'FAIL'))
   call PTRN(n,x,fcost,grad,grad_preco,optim,FLAG)
   if(FLAG.eq.'GRAD') then
      call Rosenbrock(x,fcost,grad)
      grad_preco(:)=grad(:)
   elseif(FLAG.eq.'HESS') then
      call Rosenbrock_Hess(x,optim%d,optim%Hd)
   elseif(FLAG.eq.'PREC') then
      optim%residual_preco(:)=optim%residual(:)
   endif
enddo
```

Reverse communication loop.

```
*******************************************************************************
                    PRECONDITIONED TRUNCATED NEWTON ALGORITHM
*******************************************************************************
     Convergence criterion   :    1.00E-08
     Niter_max               :      100
     Initial cost is         :    5.65E+01
     Initial norm_grad is    :    4.75E+02
     Maximum CG iter         :       5
*******************************************************************************
```

Niter	fk	||gk||	fk/f0	alpha	nls	nit_CG	eta	ngrad	nhess
0	5.65E+01	4.75E+02	1.00E+00	1.00E+00	0	0	9.00E-01	0	0
1	1.73E+00	7.19E+01	3.05E-02	1.00E+00	0	1	9.00E-01	2	1
2	6.94E-02	2.92E+00	1.23E-03	1.00E+00	0	1	8.43E-01	3	2
3	6.65E-02	1.90E-01	1.18E-03	1.00E+00	0	1	7.59E-01	4	3
4	3.51E-02	3.14E+00	6.21E-04	2.50E-01	2	2	6.40E-01	7	5
5	3.35E-02	2.36E+00	5.92E-04	2.50E-01	0	1	9.00E-01	8	6
6	3.25E-02	1.77E+00	5.76E-04	2.50E-01	0	1	8.43E-01	9	7
7	3.20E-02	1.33E+00	5.67E-04	2.50E-01	0	1	7.59E-01	10	8
8	3.17E-02	9.97E-01	5.61E-04	2.50E-01	0	1	7.52E-01	11	9
9	3.15E-02	7.49E-01	5.58E-04	2.50E-01	0	1	7.54E-01	12	10
10	3.14E-02	5.62E-01	5.57E-04	2.50E-01	0	1	7.57E-01	13	11

Output file `iterate_PTRN.dat`.

4.6.4 End of the loop and output file

At the end of the reverse communication loop, either convergence has been reached and the communication flag is 'CONV', or the linesearch has failed and the communication flag is 'FAIL'. In both cases, the vector `x` contains the last value of the minimization sequence (2.3), the best approximation to the solution of the minimization problem that can be found using PTRN.

The convergence history is written in the file `iterate_PTRN.dat`. This file contains a remainder of the optimization settings: convergence criterion and maximum number of iterations. He also presents the initial cost without normalization and the initial norm of the gradient. Then, it presents on 10 columns the convergence history.

- Column 1 is for the nonlinear iteration number.
- Column 2 is for the non normalized cost function value.
- Column 3 is for the norm of the gradient.
- Column 4 is for the relative cost function value (normalized by the initial value, on the first iteration this value is then always equal to 1).
- Column 5 is for the size of the steplength taken.
- Column 6 is for the number of linesearch iteration for determining the steplength.
- Column 7 is for the number of conjugate gradient iteration used to compute the descent direction
- Column 8 is for the forcing term η used to define the stopping criterion for the inner iterations.
- Column 9 is for the total number of gradient computation.
- Column 10 is for the total number of Hessian-vector products.

Additional information on the convergence of the inner linear systems is written in the file `iterate_PTRN_CG.dat`. This file contains a remainder of the optimization settings: convergence criterion, maximum number of iterations, maximum number of iteration for the resolution of the inner linear systems. He also presents the initial cost without normalization and the initial norm of the gradient. Then, for each nonlinear iteration, a convergence history of the inner liner system using the matrix-free conjugate gradient is presented. The number of the nonlinear iteration appears at the top, as well as the value of the forcing term η for this nonlinear iteration. Then, the convergence history is presented on 4 columns.

- Column 1 is for the iteration number
- Column 2 is for the value of the quadratic function associated with the linear system, which is supposed to be symmetric definite. This information is available only if the option `optim%debug` is set to `true`. If it is set to `false`, only 0 is written as a default value.
- Column 3 is for the norm of the residuals of the inner system.
- Column 4 is for the relative residuals value. When this value becomes lower than η, the stopping criterion is satisfied.

```
*********************************************************************************
                          TRUNCATED NEWTON ALGORITHM
                             INNER CG HISTORY
*********************************************************************************
     Convergence criterion   :    1.00E-08
     Niter_max               :      100
     Initial cost is         :    5.65E+01
     Initial norm_grad is    :    4.75E+02
     Maximum CG iter         :           5
*********************************************************************************
---------------------------------------------------------------------------------
 NONLINEAR ITERATION    0 ETA IS :      9.00E-01
---------------------------------------------------------------------------------
  Iter_CG       qk          norm_res      norm_res/||gk||
        0     0.00E+00      4.75E+02       1.00E+00
        1     0.00E+00      1.86E+01       3.92E-02
---------------------------------------------------------------------------------
 NONLINEAR ITERATION    1 ETA IS :      8.43E-01
---------------------------------------------------------------------------------
  Iter_CG       qk          norm_res      norm_res/||gk||
        0     0.00E+00      7.19E+01       1.00E+00
        1     0.00E+00      5.83E-01       8.11E-03
---------------------------------------------------------------------------------
 NONLINEAR ITERATION    2 ETA IS :      7.59E-01
---------------------------------------------------------------------------------
  Iter_CG       qk          norm_res      norm_res/||gk||
        0     0.00E+00      2.92E+00       1.00E+00
        1     0.00E+00      1.89E-01       6.49E-02
---------------------------------------------------------------------------------
 NONLINEAR ITERATION    3 ETA IS :      6.40E-01
---------------------------------------------------------------------------------
  Iter_CG       qk          norm_res      norm_res/||gk||
        0     0.00E+00      1.90E-01       1.00E+00
        1     0.00E+00      4.82E+00       2.53E+01
        2     0.00E+00      1.63E-04       8.55E-04
```

Output file `iterate_TRN.dat`.

5 Technical details

5.1 Writing intermediate values of x

It is possible to follow the construction of the sequence of iterates x_k by tracking an additional flag: 'NSTE'. When the communicator flag is equal to 'NSTE', this means that a descent direction and a steplength in this direction has been found. The unknown x has been updated, and the user may want to print it or save it in a file. This option is available for all the routines of the SEISCOPE OPTIMIZATION TOOLBOX.

Doing so only requires to add one line in the reverse communication loop. We give an example for the PSTD algorithm (it is exactly the same for the other algorithms).

```
do while ((FLAG.ne.'CONV').and.(FLAG.ne.'FAIL'))
   call PSTD(n,x,fcost,grad,optim,FLAG)
   if(FLAG.eq.'GRAD') then
      !compute cost and gradient at point x
      call Rosenbrock(x,fcost,grad)
   elseif(FLAG.eq.'NSTE') then
      write(*,*) x(:) ! or save it into disk or...
   endif
enddo
```

Tracking the sequence of iterates in the reverse communication loop.

5.2 Linesearch algorithm

5.2.1 The Wolfe criterion

The linesearch algorithm which is implemented computes a steplength α which satisfies the Wolfe criterion. For a given descent direction Δx, α should satisfy

$$f(x + \alpha\Delta x) \leq f(x) + m_1\alpha\nabla f(x)^T\Delta x, \tag{5.1}$$

and

$$\nabla f(x + \alpha\Delta x)^T\Delta x \geq m_2\nabla f(x)^T\Delta x \tag{5.2}$$

where m_1 and m_2 are scalar parameters. These parameters are set respectively to

$$m_1 = 10^{-4}, \;\; m_2 = 0.9 \tag{5.3}$$

This is done in the routines `init_method.f90`, in the variables

`optim%m1` and `optim%m2`.

The initialization of the steplength α is also done in the routines `init_method.f90`. By default, α is initialized to 1, in the variable

`optim%alpha`

5.2.2 Linesearch algorithm

The algorithm for computing a steplength α which satisfies the Wolfe conditions is as follows

1. Initialize α_{min} and α_{max} to 0.
2. Check if the current value of α satisfies the Wolfe condition. If this is the case **stop**.
3. If the first condition is not satisfied, then
   - $\alpha_{max} = \alpha$
   - $\alpha = 0.5 \times (\alpha_{min} + \alpha_{max})$
   - Go back to 2
4. If the second condition is not satisfied, then
   - $\alpha_{min} = \alpha$
   - If $\alpha_{max} = 0$ then $\alpha = 10 \times \alpha$
   - If $\alpha_{max} \neq 0$ then $\alpha = 0.5 \times (\alpha_{min} + \alpha_{max})$
   - Go back to 2
5. If no suitable steplength α has been found after `optim%nls_max` linesearch iterations, declare a linesearch failure: `FLAG` is set to 'FAIL'.

Note that each time the algorithm goes through step 2, the computation of $\nabla f(x + \alpha \Delta x)$ is required.

In addition, the parameter

`optim%nls_max`

is set in the routines `init_`*`method`*`.f90`. By default, it is set to 20 in all the optimization routines of the SEISCOPE OPTIMIZATION TOOLBOX.

5.2.3 Initialization of the linesearch parameter α

In practice, for reasonably smooth functions, the behavior of the minimization algorithms is as follows:

- At first nonlinear iteration, the linesearch algorithm can require a certain number of iterations to converge to a first steplength α which satisfies the Wolfe criterion.
- After the first nonlinear iteration, since the previous value of α is used as a first guess, none or very few linesearch iterations should be necessary (unless the cost function presents rapid variations)

The user may want to speed-up the process to avoid spending to much time in the linesearch process. To do so, it is possible to modify the initial value for the steplength to a value closer than the optimal one. This can be done by modifying the variable

`optim%alpha`

in the routines `init_`*`method`*`.f90`.

In the current version, in order to force the minimization algorithms to converge as far as they can, a special feature has also be implemented. As stated in the previous section, a linesearch failure is declared whenever no suitable steplength has been found after `otim%nls_max` linesearch iterations have been performed. However, if the steplength computed after this maximum number of linesearch iteration produces a decrease of the misfit function, it is accepted, even if it does not satisfy the Wolfe criterion. This situation however occurs very rarely to the best of our knowledge.

5.2.4 Bound constraints: projection into the feasible set

In the SEISCOPE OPTIMIZATION TOOLBOX, we implement a simple method to account for bound constraints. We want to ensure that the sequence x_k stays within the box Ω, defined as

$$\Omega = \prod_{i=1}^{n} [a_i; b_i] \subset \mathcal{R}^n, \quad n \in \mathcal{N} \tag{5.4}$$

Each time a steplength α is tested within the linesearch process, the corresponding iterate

$$x_{k+1} = x_k + \alpha \Delta x_k \tag{5.5}$$

is projected into the feasible set Ω. The Wolfe criterion are then evaluated at the points

$$\widetilde{x}_{k+1} = \mathcal{P}_\Omega x_{k+1} \tag{5.6}$$

where the component i of $\mathcal{P}_\Omega z$ is given by

$$(\mathcal{P}_\Omega z)_i = \begin{cases} z_i & \text{if} \quad a_i \leq z_i \leq b_i \\ a_i + \tau & \text{if} \quad z_i < a_i \\ b_i - \tau & \text{if} \quad z_i > b_i \end{cases} \tag{5.7}$$

The parameter τ is the tolerance `optim%threshold` set by the user.

5.2.5 Output files and debug option

When the debug option `optim%debug` is set to true, the output files generated by the SEISCOPE OPTIMIZATION TOOLBOX routines will contain the convergence history of the linesearch algorithm. At each iteration of the linesearch algorithm, step 3 of the linesearch algorithm will be identified as `failure 1`. Step 4 will be identified as `failure 2`. The different values will be printed:

- `fcost` which correspond to $f(x + \alpha_k \Delta x)$

- `optim%f0` which correspond to $f(x_0)$
- `optim%fk` which correspond to $f(x)$
- `optim%alpha` which correspond to α_k
- `optim%q0` which correspond to $\nabla f(x)^T \Delta x$
- `optim%q` which correspond to $\nabla f(x + \alpha_k \Delta x)^T \Delta x$
- `optim%m1` which corresponds to m_1
- `optim%cpt_ls` which is the counter for the current number of linesearch iterations

5.3 Nonlinear conjugate gradient

Implementation of the nonlinear conjugate gradient differs from the computation of the scalar parameter β_k from (2.5). Standard formulas are Fletcher-Reeves or Polak-Ribière formulas (see Nocedal and Wright (2006)). However, based on these formulations, the nonlinear conjugate gradient requires to satisfy the **strong** Wolfe condition to ensure global convergence toward local minima. As we wanted to use the same linesearch procedure for all the routines within the SEISCOPE OPTIMIZATION TOOLBOX, we decided to implement the nonlinear conjugate gradient algorithm proposed by Dai and Yuan (1999). This algorithm only requires the satisfaction of the standard Wolfe conditions to ensure global convergence. Following this algorithm, the scalar β_k is computed from

$$\beta_k = \frac{\nabla f(x_k)^T P_k \nabla f(x_k)}{\left(\nabla f(x_k) - \nabla f(x_{k-1})\right)^T \Delta x_{k-1}}. \tag{5.8}$$

5.4 Practical issues for the truncated Newton method

5.4.1 Choice of η_0 for the truncated Newton method

The initial forcing term η_0 controls the precision of the inner linear system resolution at the first nonlinear iteration. The value of η_0 is set in the routines `init_TRN.f90` and `init_PTRN.f90`. The default value is 0.9

```
optim%eta=0.9
```

This initial value is proposed by Eisenstat and Walker (1994). However, smaller values can be chosen to force the algorithm to solve the inner linear system more accurately at the first nonlinear iteration. This can be interesting when the function to minimize is close to a quadratic function for instance. For the Rosenbrock function, an initial value

```
optim%eta=0.1
```

will speed-up the convergence.

5.4.2 Output files and debug option

When the debug option `optim%debug` is set to `true`, the output files `iterate_TRN_CG.dat` and `iterate_PTRN_CG.dat` contain additional information on the decrease of the quadratic function which is minimized during the conjugate gradient resolution. This quadratic function is

$$q_k(\Delta x) = \Delta x^T H(x_k) \Delta x + \nabla f(x_k)^T \Delta x \tag{5.9}$$

Since the initial guess for Δx is systematically 0, at the first inner iteration, we have

$$q_k(\Delta x) = 0 \tag{5.10}$$

Then, while the Hessian matrix $H(x_k)$ remains definite positive, the quantity $q_k(\Delta x)$ decreases throughout the inner iteration of conjugate gradient. The decrease rate provides additional information on the convergence rate of the conjugate gradient. When negative eigenvalues are detected , the conjugate gradient iterations are stopped.

References

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

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

Eisenstat, S. C. and Walker, H. F. (1994). Choosing the forcing terms in an inexact Newton method. *SIAM Journal on Scientific Computing*, 17:16–32.

Métivier, L., Bretaudeau, F., Brossier, R., Operto, S., and Virieux, J. (2014). Full waveform inversion and the truncated newton method: quantitative imaging of complex subsurface structures. *Geophysical Prospecting*, In press.

Métivier, L., Brossier, R., Virieux, J., and Operto, S. (2013). Full Waveform Inversion and the truncated Newton method. *SIAM Journal On Scientific Computing*, 35(2):B401–B437.

Nocedal, J. and Wright, S. J. (2006). *Numerical Optimization*. Springer, 2nd edition.