Algorithm xxx: NOMAD: Nonlinear Optimization with the MADS algorithm

Sébastien Le Digabel [*]

August 16, 2010

Using NOMAD

This document is not the user manual of NOMAD which is included in the NOMAD package on the internet [3]. This section gives instructions to get quickly started with the optimization of a blackbox problem. Once installed, NOMAD is easy to use as most of the parameters have defaults. In fact all algorithmic parameters have defaults and for a first try, only the parameters describing the problem are necessary.

There are two modes for using NOMAD. In batch mode, the blackbox problem corresponds to a separated program that the NOMAD executable will launch via system calls and temporary files. The other mode is the library mode: The user programs and links its problem directly with the NOMAD static library (provided with the NOMAD package). The library mode is a more advanced mode and the user has to learn the NOMAD function prototypes and how to link its own code with the library. This user code has typically to be written in C++, but examples in the NOMAD package show programs using the library mode with a Windows DLL coded in Delphi, and a mini-application written in FORTRAN. For problems which are not costly to compute, the execution is much faster because no temporary files are created. For costly blackbox problems, there is no speed gain by using the library mode. Also, if the problem has hidden constraints, violating them may possibly interrupt the whole NOMAD execution, unless these violations are detected and if exceptions are thrown by the user. In batch mode, when hidden constraints are violated, and if this crashes the blackbox, then NOMAD will simply ignore the corresponding evaluations.

This section focuses on the batch mode and is based on the following academic problem taken from [6] with $n = 5$:

[*]GERAD and Département de mathématiques et de génie industriel, Ecole Polytechnique de Montréal, C.P. 6079, Succ. Centre-ville, Montréal, Québec H3C 3A7 Canada, www.gerad.ca/Sebastien.Le.Digabel, email: Sebastien.Le.Digabel@gerad.ca

$$\min_{x\in\mathbb{R}^5} f(x) = -\left| \frac{\sum_{i=1}^{5} \cos^4 x_i - 2\prod_{i=1}^{5} \cos^2 x_i}{\sqrt{\sum_{i=1}^{5} i x_i^2}} \right|$$

$$\text{subject to} \begin{cases} -\prod_{i=1}^{5} x_i + 0.75 \le 0 \\ \sum_{i=1}^{5} x_i - 37.5 \le 0 \\ \quad 0 \le x_i \le 10, \; i = 1,2,...,5. \end{cases} \tag{1}$$

Installation

NOMAD is available through three packages, for Windows, Mac, and Linux/Unix. After downloading the adequate file, the user must install it. On Windows, an installer program automatically copies the pre-compiled executables and libraries, which are then ready to use. On Mac, the files with executables and libraries are available inside a disk image. On Linux and Unix, the user has to compile the code with the `install` script. Note that the compilation of the source code is also possible on Mac and Windows, if a C++ compiler is installed. Any C++ compiler should work as NOMAD has been developed with Linux without non-standard libraries and with the `-ansi` and `-pedantic` options of `gcc`, for the scalar version. For Windows systems, NOMAD has been designed to compile with the `minGW` compiler and Microsoft Visual C++ 2008. For the parallel version, the standard level depends on which MPI implementation is used.

Once the installation step has succeeded, the user is invited to define its own environment variables to ease the access to the software and the compilation of the examples.

Defining the problem

In batch mode, the problem reduces to the blackbox executable and possibly to bounds on the variables. Bound constraints are the only constraints that may be indicated outside the blackbox. Of course the blackbox is still valid if it codes directly its own bounds, but NOMAD will be more efficient with that knowledge. There are two ways to specify the bounds: They may be directly specified in the parameters file or entered in two text files that NOMAD will load.

The blackbox executable has very few design constraints to respect: It may be coded in any language as long as it is callable from a command line. For example, a blackbox that executes itself from a graphical user interface is not valid for NOMAD. For such applications, the user must code a wrapper program that will be callable from the command line.

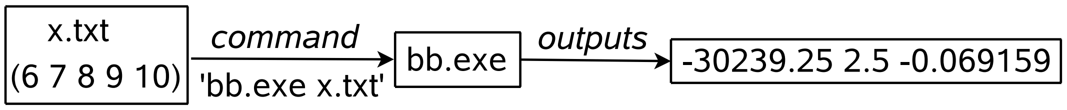

Figure 1: Example of a blackbox call sequence. The blackbox program is `bb.exe`, which is executed with the command line instruction `'bb.exe x.txt'` where `x.txt` is a text file containing a trial point coordinates. The blackbox has first to read this file and then to compute the functions defining the problem. When this is done, the function values are displayed on the standard output. The values given here are consistent with Problem (1).

More than one executable can be used. This is relevant when the first executable displays EB constraints, and if one of these constraints is violated: In that case, NOMAD will not call the other executables. We assume from now on that only one executable is used. NOMAD executes the blackbox with system calls. It indicates a trial point x at which the blackbox must compute the functions f and c_j's. The point x is given in a text file containing the coordinates separated by spaces. Figure 1 shows an example on how a blackbox should be called. The name of this file is given as an argument in the command call. The results of the functions computations have to be displayed on the standard output by the blackbox, and it is preferable to display these outputs with a good numerical precision. The choice for the display order is left to the user and will be specified in the parameters file.

If the blackbox needs to create temporary files, those should be created with unique names to be compatible with the parallel versions of NOMAD. To do so, it is possible to make NOMAD include a unique tag in the input file. Note also that if the problem has scaling issues, the user can scale the problem variables directly into the blackbox or with the `SCALING` parameter.

Creating a parameters file

NOMAD possesses several parameters for a great flexibility. However, all the algorithmic parameters have default values allowing out-of-the-box use without deep knowledge of the software and of the algorithm.

Only the most important parameters are evoked in this paper. All the parameters are detailed in the user guide, and it it also possible to display help on them directly from the NOMAD executable, by indicating a keyword after a `-h` flag. For example, entering the command `'nomad -h mesh'` will display all the parameters related to the mesh. Some other keywords such as `constraints` or `stop` are also defined to give special help.

It is indicated here how to create the basic parameters file necessary to launch NO-

MAD on Problem (1) which blackbox executable has been coded according to Section . This basic file only needs to specify the problem characteristics.

NOMAD automatically defines the problem directory as the one where the parameters file is located. All the file names correspond to files (inputs or outputs) that must be located relatively to the problem directory, though it is possible to define these locations with absolute paths. It is then adequate to create the parameters file at the same location than the blackbox program.

The file associated to Problem (1) is shown in Figure 2. The starting point in the figure is defined to be $x_0 = (6\ 7\ 8\ 9\ 10)^T$. It could also have been specified with a text file containing the coordinates separated with spaces or line breaks. Notice the use of parameter `BB_OUTPUT_TYPE` to define what are the output types of the blackbox. Here it indicates that the last value displayed at the execution of `bb.exe` is the objective value to minimize and the two first values correspond to the constraints $c_1(x) \leq 0$ and $c_2(x) \leq 0$. The keyword `CSTR` has been used, meaning that NOMAD will treat these constraints with the default PB constraints handling strategy.

Bounds may be indicated directly into the parameters file or with additional text files. For Problem (1), one line in the example of Figure 2 is sufficient to specify them.

In the example, no algorithmic parameters are defined. In particular no stopping criterion is indicated, which will let NOMAD run until the mesh size parameter Δ_k^m drops below the NOMAD default precision of 10^{-13}.

```
DIMENSION       5
BB_EXE          bb.exe
BB_OUTPUT_TYPE  CSTR CSTR OBJ
x0              ( 6  7  8  9  10 )
LOWER_BOUND     * 0.0
UPPER_BOUND     * 10.0
```

Figure 2: A basic parameters file for Problem (1), assuming that the blackbox executable created accordingly to Section is named `bb.exe` and is located in the same directory than the parameters file.

Running NOMAD

In batch mode, NOMAD is executed with the command '`nomad param.txt`' where `param.txt` is the parameters file. A stopping criteria on the maximum number of blackbox evaluations is added to the parameters file of Figure 2 to show an example of the NOMAD output on Problem (1) with a reasonable length. This criteria corresponds to the `MAX_BB_EVAL` parameter and it is set to 300 evaluations with the instruction '`MAX_BB_EVAL 300`' in the parameters file. Several stopping conditions are available in addition to the maximum number of blackbox evaluations tested here. For example, minimum mesh or poll sizes can be specified as well as a maximum time limit. NOMAD can also be interrupted when the objective value reaches a given limit or simply after a feasible solution has been found. Type the command '`nomad -h stop`' to see a complete description on all the parameters related to the stopping conditions.

```
NOMAD - version 3.4.1 - www.gerad.ca/nomad

Copyright (C) 2001-2010 {
        Mark A. Abramson     - The Boeing Company
        Charles Audet        - Ecole Polytechnique de Montreal
        Gilles Couture       - Ecole Polytechnique de Montreal
        John E. Dennis, Jr.  - Rice University
        Sebastien Le Digabel - Ecole Polytechnique de Montreal
}

Funded in part by AFOSR and Exxon Mobil.

License   : '$NOMAD_HOME/src/lgpl.txt'
User guide: '$NOMAD_HOME/doc/user_guide.pdf'
Examples  : '$NOMAD_HOME/examples'
Tools     : '$NOMAD_HOME/tools'

Please report bugs to nomad@gerad.ca

MADS run {

        BBE     OBJ

        6       -0.09090074078
        13      -0.093399105
        115     -0.1080484746
        160     -0.1187621478
        161     -0.1525259466
        174     -0.1647326914
        175     -0.194081576
        187     -0.1963317314
        218     -0.2081777291
        228     -0.2191602101
        230     -0.2195574212
        231     -0.2464497881
        283     -0.25334522
        296     -0.2601065155
        300     -0.2601065155

} end of run (max number of blackbox evaluations)

blackbox evaluations   : 300
best infeasible solution: ( 0.125 0 0 0 0 ) h=0.75 f=-24.00193288
best feasible solution  : ( 3.25 3 5.75 3 1.5 ) h=0 f=-0.2601065155
```

Figure 3: NOMAD output on Problem (1) with parameters of Figure 2.

The output of NOMAD for Problem (1), with parameters from Figure 2, is displayed in Figure 3. The amount of displayed information is dictated by the `DISPLAY_DEGREE` parameter with values ranging from 0 (no display at all) to 2 (full display). During the algorithm execution, with the default display degree, information is given every time a new feasible iterate is found. In the example of Figure 3, the displayed information is the number of blackbox evaluations and the objective value. This information is completely customizable with the `DISPLAY_STATS` parameter which recognizes special keywords such as `OBJ` for the objective value, `SOL` for point coordinates, or `TIME` for the elapsed time. Understanding the use of this parameter allows for example to directly output LaTeX entries. Notice the final solutions displayed by the execution and the fact that the best feasible point has a constraint violation h of 0 while the best infeasible point as a strictly positive value for h. In addition, the first success appears after 6 evaluations: The starting point was not feasible and it took NOMAD 6 evaluations to get its first feasible solution with a value of $\simeq -0.09$ for the objective.

It is possible to generate up to three output files: A history file, containing all the NOMAD trial points, a solution file with the final $\hat{x}$ solution (if feasible), and a configurable file containing all the successes with the same logic as the `DISPLAY_ STATS` parameter.

Finally the `CACHE_FILE` parameter may be specified to create and maintain a binary file containing all the evaluations. This file can then be used for other NOMAD runs.

Advanced use and flexibility in NOMAD

As already mentioned, there are a lot of parameters in NOMAD. The reader is invited to consult the detailed list of parameters in the user guide [3] or to explore the output of the '`nomad -h`' command.

NOMAD can minimize biobjective problems: It will then construct a list of undominated solutions which are an approximation of the Pareto front. In order to define a biobjective problem, the user must simply define two blackbox outputs with the keyword `OBJ`. NOMAD then automatically uses BiMADS and launches a series of MADS runs. There again, all the algorithmic parameters have default values and the advanced user can change them (enter '`nomad -h multi`' to display all these parameters). The most useful biobjective parameter is `MULTI_OVERALL_BB_EVAL` which is the equivalent of `MAX_BB_EVAL` for single-objective optimization. The `MAX_ BB_EVAL` parameter is still used in biobjective mode but corresponds to the maximum number of blackbox evaluations for a single MADS run.

In order to improve performance, the user may try several strategies. It would be too long to enumerate all these strategies in the present paper but here are some easy measures that can help. If NOMAD is used in batch mode on a network, the user should indicate a local directory (`/tmp` on Unix for example) where all the temporary files are created (parameter `TMP_DIR`). This way, file manipulations will be faster. This will however only improve performance for non-costly blackboxes, when the cost of reading and writing files is non-negligible compared to the cost of the evaluations.

Other parameters may have a great impact on the quality of the solution: For example the starting point (`X0`) or the value of the initial mesh size parameter (`INITIAL_MESH_SIZE`). The parameter `SNAP_TO_BOUNDS` may also change performance. It decides if trial points generated outside bounds are modified so that too large or too small coordinates are set to their bounds. This strategy is used by default but it may be necessary for some problems to disable it. The user may change the random seed used for the LT-MADS directions (`SEED`) or the Halton seed of the OrthoMADS directions (`HALTON_SEED`). The LH and VNS search steps are also available via the parameters `LH_SEARCH` and `VNS_SEARCH`. The user must though be aware of the possible additional cost in terms of evaluations. In some cases, scaling can also be a performance issue.

Additional parameters may be used to indicate a surrogate function for the blackbox. Surrogate executables must have the same outputs as the true functions they substitute and must be callable the same way. Wishfully, they are also cheap to execute.

The library mode of NOMAD offers even more flexibility and performance: Several functionalities can be redefined via the virtual functions of NOMAD, but this mode is only accessible with a minimal knowledge of C++ and object oriented programming.

In library mode the user must create a main function and link it with the NOMAD static library file `nomad.a` (on Linux). The blackbox functions may be defined inside the code or as separated programs as in the batch mode. A `NOMAD::Parameters` object must be constructed with the `set` methods included in the class interface. Once the problem and the parameters are defined, a `NOMAD::Mads` object must be created and executed with `NOMAD::Mads::run()` or `NOMAD::Mads::multi_run()` in the biobjective case. Optimization results are accessible from `NOMAD::Mads` and `NOMAD::Stats` classes.

Here is a short list of functionalities available with the library mode: users can preprocess the trial points before they are evaluated and give the evaluation points a priority for being evaluated. Virtual customizable functions are also automatically called after each iteration and after each success (functions `update_iteration()` and `update_success()` from `NOMAD::Evaluator`). Users can also code their own search strategy. To do so, they must design a subclass of the `NOMAD::Search` class and link an object of this class to the `NOMAD::Mads` object used to run the algorithm. Virtual functions are also necessary to treat problems with categorical variables as they allow the user to define the neighborhoods of these variables.

For all these virtual functions, the user must be aware that modifying or generating new trial points may destroy the global convergence properties of the algorithm, as in the example 2.2 of [1] where a user-defined search leads to a point without any optimality property. The main condition to ensure in any modification is that points remain on the mesh. The function `NOMAD::Point::project_to_mesh()` is available for that purpose. The decision to project points to the mesh is explicitely left to the user for a greater flexibility.

Several examples are given in the NOMAD package, illustrating various advanced situations in batch or library modes. For example, a multi-start based on LH sampling is given as well as a user search strategy. Other examples show that NOMAD can be interfaced with blackbox problems that are coded in other languages: Such examples include AMPL [4], MATLAB, GAMS [2], CUTEr [5], and an example where the blackbox is a Windows dynamic library (DLL). Another example shows how to use NOMAD version 3 with the previous version 2.

References

[1] C. Audet and S. Le Digabel. The mesh adaptive direct search algorithm for periodic variables. Technical Report G-2009-23, Les cahiers du GERAD, 2009.

[2] A. Brooke, D. Kendrick, and A. Meeraus. *GAMS: A Users' Guide*. The Scientific Press, Danvers, Massachusetts, 1988.

[3] S. Le Digabel. NOMAD user guide. Technical Report G-2009-37, Les cahiers du GERAD, 2009.

[4] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Thomson/Brooks/Cole, Pacific Grove, California, second edition, 2003.

[5] N. I. M. Gould, D. Orban, and Ph. L. Toint. CUTEr (and SifDec): a constrained and unconstrained testing environment, revisited. *ACM Transactions on Mathematical Software*, 29(4):373–394, 2003.

[6] A. Hedar and M. Fukushima. Derivative-free filter simulated annealing method for constrained continuous global optimization. *Journal of Global Optimization*, 35(4):521–549, 2006.