

NOMAD user guide

version 3.0

Sébastien Le Digabel

December 3, 2008

Contents

1	Introduction	2
2	Installation	3
2.1	Pre-compiled executables	3
2.2	Compile the source code	3
2.2.1	Linux / Unix / Mac OS X	4
2.2.2	Windows with <code>minGW</code>	4
2.2.3	Windows with <code>Visual C++</code>	4
2.2.4	Library compilation	4
3	Basic use of the NOMAD batch mode	5
3.1	Creation of a basic parameters file	5
3.2	Basic instructions on black-box programs	6
4	Use of the NOMAD library	11
4.1	Definition of the problem	11
4.2	The main function	14
4.2.1	Parameters	14
4.2.2	Evaluator declaration and algorithm run	14
4.2.3	Access to the solution and to optimization data	15
4.2.4	Multiple runs	15
5	Future versions	18
5.1	Algorithm future developments	18
5.2	Other programs or algorithms included in the NOMAD package	18
5.3	Improved documentation	18
	Bibliography	22

1 Introduction

NOMAD is a C++ implementation of the Mesh Adaptive Direct Search (MADS) algorithm [6, 17, 19], designed for constrained optimization of black-box functions in the form

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

where $\Omega = \{x \in X : c_j(x) \leq 0, j \in J\} \subset \mathbb{R}^n$, $f, c_j : X \rightarrow \mathbb{R} \cup \{\infty\}$ for all $j \in J = \{1, 2, \dots, m\}$, and where X is a subset of \mathbb{R}^n .

Developers of the method behind NOMAD include

- Mark A. Abramson (Mark.A.Abramson@boeing.com), The Boeing Company.
- Charles Audet (www.gerad.ca/Charles.Audet), GERAD and Département de mathématiques et de génie industriel, École Polytechnique de Montréal.
- J.E. Dennis Jr. (www.caam.rice.edu/~dennis), Computational and Applied Mathematics Department, Rice University.
- Sébastien Le Digabel (www.gerad.ca/Sébastien.Le.Digabel), GERAD and Département de mathématiques et de génie industriel, École Polytechnique de Montréal.

Programmers are Sébastien Le Digabel for version 3 and [Gilles Couture](#) (GERAD) for previous versions.

NOMAD is designed to be used in two different modes: batch and library. The batch mode is intended for a basic and simple usage of the MADS method, while the library mode allows more flexibility. For example, in batch mode, users must define their separate black-box program, that will be called with system calls by NOMAD. In library mode, users can define their black-box function as C++ code that will be directly called by NOMAD, without system calls and temporary files. This document explains how to get started with the batch mode in Section 3, and with the library mode in Section 4. Future additions to the documentation are detailed in Section 5.3.

NOMAD should be cited with reference [4]. Other relevant papers by the developers are accessible through the NOMAD web site www.gerad.ca/nomad.

The project started in 2001, and was funded in part by AFOSR, CRIAQ, FQRNT, LANL, NSERC, the Boeing Company, and ExxonMobil Upstream Research Company.

2 Installation

NOMAD is developed under linux with the gcc C++ compiler (g++), versions 3 and 4. It also has been tested on Unix, Mac OS X Leopard with Xcode (gcc 4), Windows XP with minGW (gcc for Windows), and visual C++ .net. NOMAD is freely distributed under the GNU General Public License, that can be read in the file `gpl-3.0.txt` provided by the package, or at www.gnu.org/licenses.

There are two ways of using NOMAD, one can directly use an executable or compile the source code. A .zip archive containing the NOMAD package is available for download on the web site. The .zip file contains the directory structure described by Figure 1, where `$NOMAD_HOME` corresponds to the base directory extracted from the archive.

```
$NOMAD_HOME
|- bin
|- doc
|- examples
|   |- batch_mode
|   |   |- advanced
|   |   |- basic
|   |- library_mode
|       |- example1
|       |- example2
|- lib
|- src
```

Figure 1: Directory structure of the NOMAD package.

2.1 Pre-compiled executables

NOMAD batch mode executables are available on the web site, for several platforms. They are located in directory `$NOMAD_HOME/bin`. In order to avoid compiling the code, you can simply use the executable corresponding to your system. You may erase the ones not designed for your system.

2.2 Compile the source code

If no executable is available for your platform, or if you want to use the library mode of NOMAD, then you need to compile the source code, located in `$NOMAD_HOME/src`.

2.2.1 Linux / Unix / Mac OS X

From a terminal opened in directory `$NOMAD_HOME`, type `'make release'`. This will create the executable `nomad` located in `$NOMAD_HOME/bin`.

2.2.2 Windows with minGW

Same procedure as in 2.2.1 except that, in the makefile, you must replace `GCC_X` with `GCC_WINDOWS`. The executable is `$NOMAD_HOME\bin\nomad.exe`.

2.2.3 Windows with Visual C++

Create a new console, empty project. Choose a name for your project (for example, `'project_name'`), and create the project in `$NOMAD_HOME`. Then, add all `.cpp` and `.hpp` source files to the project, and compile in `release` mode. This generates the executable file `$NOMAD_HOME\project_name\Release\project_name.exe`, which can be copied in `$NOMAD_HOME\bin` for convenience and staying consistent with this document.

2.2.4 Library compilation

If you intend to use NOMAD in library mode, you must compile the library. To do so with the makefile in the source directory, type `'make lib'`, and it will generate the file `$NOMAD_HOME/lib/nomad.a`. With `visual C++`, create an empty static library project, insert all files except `nomad.cpp`, and compile. Use the NOMAD library mode is not yet described in this document.

For correct compilation of programs that will use the NOMAD library, define the environment variable `$NOMAD_HOME` to the path where NOMAD is installed. For example, if you use a `csch` shell, insert the following line in your `.cschrc` file: `'setenv NOMAD_HOME /home/your_login/NOMAD.3.0.0'`.

3 Basic use of the NOMAD batch mode

This section explains how to get started with the NOMAD batch mode, and it gives all the steps to solve a black-box problem. The NOMAD batch mode is launched with one argument that corresponds to the name of a parameters text file, and your black-box problem has to be coded as a separated program. The different steps are:

1. Compile NOMAD or directly take a compatible executable (follow instructions of Section 2).
2. Create a directory for your problem. In this document, we use the notation `$PB_DIR` in order to refer to this directory.
3. Create your problem's black-box, which corresponds to an executable located in `$PB_DIR` (see Section 3.2).
4. Create a parameters file, for example `$PB_DIR/param.txt`, located in the problem directory (see Section 3.1). This file describes where NOMAD will find your problem and what parameters to use.
5. If the NOMAD executable corresponds to the file `$NOMAD_HOME/bin/nomad`, launch the algorithm with `'$NOMAD_HOME/bin/nomad $PB_DIR/param.txt'`.

Advanced usage of NOMAD is not described in this section. However, all parameters are described in `$NOMAD_HOME/doc/parameters.description.txt`, and an advanced example is given in `$NOMAD_HOME/examples/batch_mode/advanced`.

3.1 Creation of a basic parameters file

The parameters file is a text file given as argument to the NOMAD executable with the command `'$NOMAD_HOME/bin/nomad $PB_DIR/param.txt'`, where `param.txt` is the parameters file (which has to be located in the problem directory), and `nomad` the NOMAD executable.

For a basic usage, these parameters have to be defined:

- The number of variables (`DIMENSION`).
- The name of the black-box executable (`BB_EXE`).
- The number of outputs of the black-box executable: objective and constraints (`BB_OUTPUT_TYPE`).
- A starting point (`X0`).
- Some stopping criteria (`MAX_BB_EVAL`, for example).

Bounds on variables are defined with the `LOWER_BOUND` and `UPPER_BOUND` parameters. If no stopping criteria is specified, the algorithm will stop as soon as the mesh size reaches a certain epsilon.

An example is given in Figure 2 that corresponds to the parameters file located in `$NOMAD_HOME/examples/batch_mode/basic`. Note that all the entries of a line are ignored after the character `'#'`. The order in which the parameters appear, or their case, is unimportant.

The two constraints defined in the parameters file of Figure 2 are of different type. The first constraint $c_1(x) \leq 0$ is treated by the progressive barrier approach (PB), which allows constraint violations. The second constraint, $c_2(x) \leq 0$, is treated by the extreme barrier approach (EB) that forbids violations.

See the text file `parameters.description.txt` located in `$NOMAD_HOME/doc` for the detailed description of all parameters.

3.2 Basic instructions on black-box programs

With the batch use of NOMAD, the black-box defining your problem corresponds to a program that will be system-called by the algorithm. It can be coded in any language (even scripts), but has to respect certain conditions. It has to be callable in batch mode, as follows: If the black-box executable is `$PB_DIR/bb.exe`, one can call it with the command `'$PB_DIR/bb.exe x.txt'`. Here `x.txt` is a text file containing a total of `DIMENSION` values consisting of one value for each variable. They may be separated by spaces or line breaks.

The black-box program returns its values by displaying them in the standard output. The number of values displayed by the black-box program corresponds to the number of constraints plus one value representing the objective function value that one seeks to minimize. The constraints values correspond to constraints of the form $c_j \leq 0$ (for example, the constraint $0 \leq x_1 + x_2 \leq 10$ will have to be displayed with the two quantities $c_1(x) = -x_1 - x_2$ and $c_2(x) = x_1 + x_2 - 10$). The order of the displayed outputs corresponds to the order defined in the parameters file with parameters `BB_EXE` and `BB_OUTPUT_TYPE`. If variables have bound constraints, these are defined in the parameters file with parameters `LOWER_BOUND` and `UPPER_BOUND`. Bounds should not appear in the black-box code.

In basic mode, your black-box program cannot display other data than the objective and constraint values, but the advanced mode allows it to do so. Your code can generate temporary files, but it is preferable to include tag numbers to the file names, because future NOMAD versions will include parallelism (the advanced mode allows to include these tags in the black-box input files). If you already have a black-box program in a certain format, you need to interface it with a wrapper program

in order to match the NOMAD specifications. If your black-box program crashes in batch mode, it will not affect NOMAD: The argument that caused this crash will simply be tagged as a black-box failure.

A basic C++ program example is given in Figure 3, for the following problem with 5 variables and 2 constraints:

$$\begin{array}{ll} \min_{x \in \mathbb{R}^n} f(x) = x_5 & \\ \text{subject to} & \left\{ \begin{array}{ll} c_1(x) = \sum_{i=1}^5 (x_i - 1)^2 - 25 & \leq 0 \\ c_2(x) = 25 - \sum_{i=1}^5 (x_i + 1)^2 & \leq 0 \\ x_i & \geq -6 \quad i = 1, 2, \dots, 5 \\ x_1 & \leq 5 \\ x_2 & \leq 6 \\ x_3 & \leq 7 \end{array} \right. \end{array}$$

With gcc, you can compile the example program of Figure 3 with '`g++ -o bb.exe bb.cpp`', and test it with the text file `x.txt` containing '`0 0 0 0 0`', with the command '`bb.exe x.txt`'. This should display '`0 -20 20`', which means that the point $x = (0 \ 0 \ 0 \ 0 \ 0)^T$ has an objective value of $f(x) = 0$, but is not feasible, since the second constraint is violated ($c_2(x) = 20 > 0$).

NOMAD is flexible enough that black-box codes can be coded differently and with more sophistication in the advanced mode.

Figure 4 shows the display that the execution of NOMAD produces for the black-box program of Figure 3, with parameters file of Figure 2. Notice that the first feasible point has been found after 29 black-box evaluations. In this case, the starting point $x = (0 \ 0 \ 0 \ 0 \ 0)^T$ violates the second constraint, treated by the extreme barrier approach. In such a situation, NOMAD launches a phase one step, during which the value of the constraint violation is minimized. Once a feasible point is generated with this phase one, the original objective function is considered again.

DIMENSION	5	# number of variables
BB_EXE	bb.exe	# 'bb.exe' is a program that
BB_OUTPUT_TYPE	OBJ PB EB	# takes in argument the name of
		# a text file containing 5
		# values, and that displays 3
		# values that correspond to the
		# objective function value (OBJ),
		# and two constraints values g1
		# and g2 with form $g1 \leq 0$ and
		# $g2 \leq 0$; 'PB' and 'EB'
		# correspond to constraints that
		# are treated by the Progressive
		# or Extreme Barrier approaches
		# (all constraint handling
		# options are described in the
		# detailed parameters list)
X0	(0 0 0 0 0)	# starting point
LOWER_BOUND	* -6	# all variables are ≥ -6
UPPER_BOUND	(5 6 7 - -)	# $x_1 \leq 5$, $x_2 \leq 6$, $x_3 \leq 7$
		# x_4 and x_5 have no bounds
MAX_BB_EVAL	100	# the algorithm terminates when
		# 100 black-box evaluations have
		# been made
TMP_DIR	/tmp	# indicates a repertory where
		# temporary files are put
		# (increases performance by ~100%
		# if you're working on a network
		# account and if TMP_DIR is on a
		# local disk).

Figure 2: Example of a basic parameters file. All parameters are detailed in file `$NOMAD_HOME/doc/parameters.description.txt`.


```

#include <cmath>
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;

int main ( int argc , char ** argv ) {

    double f = 1e20, c1 = 1e20 , c2 = 1e20;
    double x[5];

    if ( argc >= 2 ) {
        c1 = 0.0 , c2 = 0.0;
        ifstream in ( argv[1] );
        for ( int i = 0 ; i < 5 ; i++ ) {
            in >> x[i];
            c1 += pow ( x[i]-1 , 2 );
            c2 += pow ( x[i]+1 , 2 );
        }
        f = x[4];
        if ( in.fail() )
            f = c1 = c2 = 1e20;
        else {
            c1 = c1 - 25;
            c2 = 25 - c2;
        }
        in.close();
    }
    cout << f << " " << c1 << " " << c2 << endl;
    return 0;
}

```

Figure 3: Example of a basic black-box program.

```

NOMAD - Nonsmooth Optimization by Mesh Adaptive Direct search
- version 3.0.0

Copyright (C) 2001-2008
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

begin of Mads::run()

    stats:
    BBE      OBJ

    29        3.875
    41        3.75
    78        3.74219
    79        3.71875
    80        3.625
    81        3.25
    87        2.25
    88        -0.75
    100       -0.75

end of Mads::run(), max number of black-box evaluations reached

black-box evaluations : 100
best infeasible point : (0.825 -1.2 -1.625 -1.25 -6) h=40.8238 f=-6
best feasible point   : (0.825 3.6 -1.625 -1.25 -0.75) h=0 f=-0.75

```

Figure 4: Output given by NOMAD on the black-box problem coded in Figure 3, with parameters file of Figure 2.

4 Use of the NOMAD library

This section explains how to create a C++ program able to call the NOMAD routines, using the pre-compiled NOMAD static library. We suppose that the library has been generated following the instructions of Section 2.2.4 and that the environment variable `$NOMAD_HOME` has been defined. Explanations are given for `linux` and `g++`, but are similar for `windows` and `visual C++`. A basic knowledge of object oriented programming with C++ is assumed.

The use of the standard C++ types for reals and vectors is of course allowed within your code, but it is suggested that you use the NOMAD types as much as possible. For reals, NOMAD uses the class `NOMAD::Double`, and for vectors, the class `Point`. A lot of functionalities have been coded for these classes, visible in files `Double.hpp` and `Point.hpp`. All the NOMAD class files are named like the classes and are located in directory `$NOMAD_HOME/src`. Other NOMAD types (essentially enumeration types) are also defined in `defines.hpp`. Some utility functions on these types can be found in `utils.hpp`.

The example shown in this section corresponds to files located in directory `$NOMAD_HOME/examples/library_mode/example1`. It is identical to the example shown in Section 3, except that no temporary files are used, and no system calls are made. Also, each compilation of the program is faster, as the NOMAD code is already compiled in the library file. For this example, just one C++ source file is used, but there could be a lot more. Other examples can be found in directory `$NOMAD_HOME/examples/library_mode` and in `$NOMAD_HOME/src/nomad.cpp`, containing the main function of the NOMAD package: this file implements the NOMAD batch mode, and could have been compiled in library mode. This illustrates the fact that even in library mode, a parameters file can be used, and system calls performed.

In a first step, a makefile has to be created in the directory where your source code is located. An example of such a makefile is shown on Figure 5. Notice that each line after `':'` has to begin with a tabulation.

We now describe the other steps for the creation of the source file `example1.cpp`, which include the header file `Mads.hpp`, and which is divided into two parts: a class for the description of the problem, and the main function. Once compiled with the makefile (type `'make'`), the binary file `example1` is created and can be executed.

4.1 Definition of the problem

Describing the black-box problem directly into the code that calls NOMAD avoids the use of temporary files and system calls by the algorithm. This is achieved by defining a derived class `My_Evaluator`, that inherits from the class `Evaluator` (see header file `Evaluator.hpp`). An example of such a class is shown in Figure 7.

The objective of this user class is to redefine the virtual method `eval_x`, that will be automatically called by the algorithm. The prototype of `eval_x` is given in

```

EXE          = example1
COMPILATOR   = g++
COMP_OPT     = -ansi -Wall -O3
L1           = $(NOMAD_HOME)/lib/nomad.a
LIBS         = $(L1) -lc -lm
INCLUDE      = -I$(NOMAD_HOME)/src -I.
COMPILE      = $(COMPILATOR) $(COMP_OPT) $(INCLUDE) -c
OBJS         = example1.o

$(EXE): $(OBJS)
    $(COMPILATOR) -o $(EXE) $(OBJS) $(LIBS) $(COMP_OPT)

example1.o: example1.cpp $(L1)
    $(COMPILE) example1.cpp

clean:
    @echo "    cleaning obj files"
    @rm -f $(OBJS)

```

Figure 5: Example of a makefile for a single C++ file linked with the NOMAD library.

Figure 6.

```

bool eval_x ( Eval_Point      & x          ,
              const NOMAD::Double & h_max   ,
              bool             & count_eval ) const;

```

Figure 6: Protoype of method `Evaluator::eval_x()`.

The argument `x` (in/out) corresponds to an evaluation point, i.e. a vector containing the coordinates of the point to be evaluated, and also the outputs of the evaluation. The coordinates are accessed with the operator `[]` (`x[0]` for the first coordinate), and outputs are set by the method `set_bb_output` (`x.set_bb_output[0]` to set the objective function value, if the objective has been defined to be the first output). Constraints have to be represented by values c_j for a constraint $c_j \leq 0$. Please refer to files `Eval_Point.hpp` and `Point.hpp` for details about the classes defining NOMAD vectors.

The second argument, the real `h_max` (in), corresponds to the current value of the barrier h_{max} parameter. It is not used in this example, but it can be used in order to interrupt an expensive evaluation, if the constraint violation value h can be quickly computed larger than h_{max} (see [19] for the definition of h and h_{max} and of

the progressive barrier method for handling constraints).

The third argument, `count_eval` (out), needs to be set to `true` if the evaluation counts as a black-box evaluation, and `false` otherwise (for example, if the user can interrupt an evaluation with the h_{max} criterion before it costs some expensive computations, then set `count_eval` to `false`).

Finally, `eval_x` should return `true` if the evaluation succeeded, and `false` if the evaluation failed.

```
class My_Evaluator : public Evaluator {
public:
    My_Evaluator ( const Parameters & p ) :
        Evaluator ( p , cout ) {}

    ~My_Evaluator ( void ) {}

    bool eval_x ( Eval_Point          & x
                  , const NOMAD::Double & h_max
                  , bool               & count_eval ) const {
        NOMAD::Double c1 = 0.0 , c2 = 0.0;
        for ( int i = 0 ; i < 5 ; i++ ) {
            c1 += (x[i]-1).pow2();
            c2 += (x[i]+1).pow2();
        }
        x.set_bb_output ( 0 , x[4] ); // objective value
        x.set_bb_output ( 1 , c1-25 ); // constraint 1
        x.set_bb_output ( 2 , 25-c2 ); // constraint 2

        count_eval = true; // count a black-box evaluation
        return true;       // the evaluation succeeded
    }
};
```

Figure 7: Example of a user class defining a hardcoded black-box problem.

Of course, more elaborated `Evaluator` subclasses can be designed in order to consider some additional problem-related parameters. Such an example can be found in `$NOMAD_HOME/examples/library_mode/example2`, where some weights are defined to change the objective function of the problem between successive optimizations (this example is taken from [23]).

The virtual method `update_success` can also be redefined in subclasses deriving from `Evaluator`. This method will be automatically invoked every time a new

success is made. The prototype of the method is `void update_success (void)`, i.e. it takes no argument and returns no data.

Another virtual method is defined in the class `Evaluator` is `compute_f()`. This method allows the user to compute the value of the objective function directly from the black-box outputs. An utilization of this method is illustrated in the example located in `$NOMAD_HOME/examples/library_mode/example2`.

4.2 The main function

Once your problem has been defined, the main function can be written. NOMAD routines can throw C++ exceptions, so it is recommended that you put your code into a `try` block.

4.2.1 Parameters

First, a `Parameters` object has to be declared. Then, parameters are defined similarly as in batch mode: each parameter `PNAME` is set with the method `set_PNAME` of the class `Parameters`. In order to see all the options, please refer to the detailed list of parameters in `$NOMAD_HOME/doc` and to the header file `Parameters.hpp`. Non-standard C++ types necessary for the call to `Parameters` set functions can be found in file `defines.hpp`. An example is given in Figure 8. This example is taken from file `example1.cpp` located in `$NOMAD_HOME/examples/library_mode/example1` and corresponds to the same parameters file example shown in Figure 2, except that no problem executable is used.

No parameters file is needed anymore, but it is possible to take the parameters from such a file, with the `Parameters`'s method `read("param.txt")` where `param.txt` is a valid parameters file. If a directory path is included in the name of the file, this path will be considered as the problem's path instead of the default location `'.'`. To display and check the parameters described by a `Parameters` object `p`, use the instruction `cout << p << endl;`.

Finally, once that all parameters have been set, the method `Parameters::check()` must be invoked in order to validate the parameters. The algorithm will not run with a non-checked `Parameters` object. If parameters are changed, `check()` must be invoked again before a new run can be executed.

4.2.2 Evaluator declaration and algorithm run

The MADS algorithm is implemented with the `Mads` class. Objects of this class are created with a `Parameters` object and an `Evaluator` object. In the example described here, the `Evaluator` object corresponds to an object of type `My_Evaluator`. A `NULL` pointer can also be used instead of the `Evaluator` object: in this case, the default evaluator will be used. Assuming that the parameter `BB_EXE` has been defined,

this default evaluator consists in evaluating the objective function via a separated black-box program and system calls.

Once that the **Mads** object is declared, run the algorithm with **Mads::run()**. An example is shown in Figure 9.

4.2.3 Access to the solution and to optimization data

In the example in `$NOMAD_HOME/examples/library_mode/example1`, final information is displayed simply with a call to the operator `<<` of the **Mads** class. However, more specialized access to solution and optimization data is allowed. To access the best feasible and infeasible points, use the methods **Mads::get_best_feasible()** and **Mads::get_best_infeasible()**. To access optimization data or algorithm statistics, call the method **Mads::get_stats()** which returns access to a **Stats** object. Then, use the access methods defined in **Stats.hpp**. For example, to display the number of black-box evaluations, write:

```
cout << "bb eval = " << mads.get_stats().get_bb_eval() << endl;
```

4.2.4 Multiple runs

The method **Mads::run()** can be invoked more than once, for multiple runs of the MADS algorithm.

A first solution for doing that is simply to declare the **Mads** object, as in Figure 10. But, in this case, the cache, containing all points from the first run, will be erased between the runs (since its it created and destructed with **Mads** objects).

A better solution consists in using the **Mads::reset()** method between the two runs and to keep the **Mads** in a more global scope. The method takes two Booleans arguments (set to **false** by default), **keep_barrier** and **keep_stats**, indicating if the barrier and statistics have to be reseted between the two runs. An example is shown in Figure 11.

Another example showing multiple MADS runs is described by the files located in `$NOMAD_HOME/examples/library_mode/example2`, for a problem taken from [23]. Note the usage of the different virtual functions, allowing to use an unique cache for the same problem parametrized differently.

```

// parameters creation:
Parameters p ( cout );

p.set_DIMENSION (5);           // number of variables

vector<bb_output_type> bbot (3); // definition of
bbot[0] = _OBJ_;                // output types
bbot[1] = _PB_;
bbot[2] = _EB_;
p.set_BB_OUTPUT_TYPE ( bbot );

p.set_X0 ( Point ( 5 , 0.0 ) ); // starting point

p.set_LOWER_BOUND ( Point ( 5 , -6.0 ) ); // all var. >= -6
Point ub ( 5 );                  // x_4 and x_5 have no bounds
ub[0] = 5.0;                    // x_1 <= 5
ub[1] = 6.0;                    // x_2 <= 6
ub[2] = 7.0;                    // x_3 <= 7
p.set_UPPER_BOUND ( ub );

p.set_MAX_BB_EVAL (100);        // the algorithm terminates
                                // after 100 bb evaluations

p.set_TMP_DIR ( "/tmp" );       // repertory for
                                // temporary files

// parameters validation:
p.check();

```

Figure 8: Example of parameters creation in library mode.

```

// custom evaluator creation:
My_Evaluator ev ( p );

// algorithm creation and execution:
Mads mads ( p , &ev , cout );
mads.run();

```

Figure 9: Evaluator and Mads objects usage.


```

{
    Mads mads ( p , &ev , cout );

    // run #1:
    mads.run();
}
{
    Mads mads ( p , &ev , cout );

    // run #2:
    mads.run();
}

```

Figure 10: Two runs of MADS with a **Mads** object at local scope. The cache is erased between the two runs.

```

Mads mads ( p , &ev , cout );

// run #1:
mads.run();

mads.reset();

// run #2:
mads.run();

```

Figure 11: Two runs of MADS with a **Mads** object at a more global scope. The cache is kept between the two runs.

5 Future versions

5.1 Algorithm future developments

- Use of simplex gradients [29, 30].
- Groups of variables.
- Generic polling directions.
- Generic search step.
- Addition of the VNS search strategy [12].
- Synchronous parallel version (PMADS)
- Categorical variables [1, 8, 9, 15, 35].
- Dynamic surrogates.
- Compatibility with a wider range of C++ environments.

5.2 Other programs or algorithms included in the NOMAD package

- `cache_manager`: program to manipulate NOMAD cache files.
- COOP-MADS: several MADS instances launched in parallel with a cache server.
- PSD-MADS: parallel space decomposition of MADS [21].
- MULTI-MADS: multi-objective variant of MADS [23].

5.3 Improved documentation

A more complete user guide will be released along with future versions, and more interactive documentation will be added directly on the [NOMAD web site](#). More precisely, future documentation additions will include:

- A NOMAD manual page (batch command).
- A problem page: this page will be inserted in the NOMAD web page and will include a collection of test problems.
- A detailed list of NOMAD actual functionalities (user guide and web page).
- A complete list of parameters (user guide and web page).

- A documentation on the NOMAD advanced batch mode (user guide and web page).
- A documentation on the NOMAD C++ library usage (web page).

Related publications by the developers

- [1] M. A. Abramson. Mixed variable optimization of a load-bearing thermal insulation system using a filter pattern search algorithm. *Optimization and Engineering*, 5(2):157–177, 2004.
- [2] M. A. Abramson. Second-order behavior of pattern search. *SIAM Journal on Optimization*, 16(2):315–330, 2005.
- [3] M. A. Abramson and C. Audet. Convergence of mesh adaptive direct search to second-order stationary points. *SIAM Journal on Optimization*, 17(2):606–619, 2006.
- [4] M. A. Abramson, C. Audet, G. Couture, J. E. Dennis, Jr., and S. Le Digabel. The NOMAD project. Software available at www.gerad.ca/nomad.
- [5] M. A. Abramson, C. Audet, and J. E. Dennis, Jr. Generalized pattern searches with derivative information. *Mathematical Programming*, Series B, 100:3–25, 2004.
- [6] M. A. Abramson, C. Audet, J. E. Dennis, Jr., and S. Le Digabel. OrthoMADS: A deterministic MADS instance with orthogonal directions. Les Cahiers du GERAD G-2008-15, GERAD, February 2008.
- [7] M. A. Abramson, O. A. Brezhneva, J. E. Dennis Jr., and R. L. Pingel. Pattern search in the presence of degeneracy. 2007, to appear.
- [8] M.A. Abramson, C. Audet, J.W. Chrissis, and J.G. Walston. Mesh adaptive direct search algorithms for mixed variable optimization. *Optimization Letters*, 3(1):35–47, January 2009.
- [9] M.A. Abramson, C. Audet, and J. E. Dennis, Jr. Filter pattern search algorithms for mixed variable constrained optimization problems. *Pacific Journal on Optimization*, 3(3):477–500, 2007.
- [10] C. Audet. Convergence Results for Pattern Search Algorithms are Tight. *Optimization and Engineering*, 5(2):101–122, 2004.
- [11] C. Audet, V. Béchar, and J. Chaouki. Spent potliner treatment process optimization using a MADS algorithm. *Optimization and Engineering*, 9(2):143–160, 2007.

- [12] C. Audet, V. Bécard, and S. Le Digabel. Nonsmooth optimization through mesh adaptive direct search and variable neighborhood search. *Journal of Global Optimization*, 41(2):299–318, June 2008.
- [13] C. Audet, A. J. Booker, J. E. Dennis, Jr., P. D. Frank, and D. W. Moore. A surrogate-model-based method for constrained optimization. Presented at the 8th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, 2000.
- [14] C. Audet, A. L. Custódio, and J. E. Dennis, Jr. Erratum: Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization*, 18(4):1501–1503, 2008.
- [15] C. Audet and J. E. Dennis, Jr. Pattern search algorithms for mixed variable programming. *SIAM Journal on Optimization*, 11(3):573–594, 2000.
- [16] C. Audet and J. E. Dennis, Jr. Analysis of generalized pattern searches. *SIAM Journal on Optimization*, 13(3):889–903, 2003.
- [17] C. Audet and J. E. Dennis, Jr. Mesh Adaptive Direct Search Algorithms for Constrained Optimization. *SIAM Journal on Optimization*, 17(1):188–217, 2006.
- [18] C. Audet and J. E. Dennis, Jr. Nonlinear programming by mesh adaptive direct searches. *SIAG/Optimization Views-and-News*, 17(1):2–11, 2006.
- [19] C. Audet and J. E. Dennis, Jr. A MADS algorithm with a progressive barrier for derivative-free nonlinear programming. Les Cahiers du GERAD G-2007-37, GERAD, May 2007.
- [20] C. Audet, J. E. Dennis, Jr., and S. Le Digabel. Globalization strategies for mesh adaptive direct search. Les Cahiers du GERAD G-2008-74, GERAD, November 2008.
- [21] C. Audet, J. E. Dennis, Jr., and S. Le Digabel. Parallel space decomposition of the mesh adaptive direct search algorithm. *SIAM Journal on Optimization*, 19(3):1150–1170, 2008.
- [22] C. Audet and D. Orban. Finding optimal algorithmic parameters using the mesh adaptive direct search algorithm. *SIAM Journal on Optimization*, 17(3):642–664, 2006.
- [23] C. Audet, G. Savard, and W. Zghal. Multiobjective Optimization Through a Series of Single-Objective Formulations. *SIAM Journal on Optimization*, 19(1):188–210, 2008.

- [24] A. J. Booker, E. J. Cramer, P. D. Frank, J. M. Gablonsky, and J. E. Dennis, Jr. Movars: Multidisciplinary optimization via adaptive response surfaces. AIAA Paper 2007–1927, Presented at the 48th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, Honolulu, 2007.
- [25] A. J. Booker, J. E. Dennis, Jr., P. D. Frank, D. W. Moore, and D. B. Serafini. Managing surrogate objectives to optimize a helicopter rotor design – further experiments. AIAA Paper 1998–4717, Presented at the 8th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, St. Louis, 1998.
- [26] A. J. Booker, J. E. Dennis, Jr., P. D. Frank, D. B. Serafini, and V. Torczon. Optimization using surrogate objectives on a helicopter test example. In J. Borggaard, J. Burns, E. Cliff, and S. Schreck, editors, *Optimal Design and Control*, Progress in Systems and Control Theory, pages 49–58, Cambridge, Massachusetts, 1998. Birkhäuser.
- [27] A. J. Booker, J. E. Dennis, Jr., P. D. Frank, D. B. Serafini, V. Torczon, and M. W. Trosset. A rigorous framework for optimization of expensive functions by surrogates. *Structural Optimization*, 17(1):1–13, February 1999.
- [28] E. J. Cramer, J. E. Dennis, Jr., P. D. Frank, R. M. Lewis, and G. R. Shubin. Problem formulation for multidisciplinary optimization. In *AIAA Symposium on Multidisciplinary Design Optimization*, September 1993.
- [29] A. L. Custódio, J. E. Dennis, Jr., and L. N. Vicente. Using simplex gradients of nonsmooth functions in direct search methods. *IMA Journal of Numerical Analysis*, 28(4):770–784, 2008.
- [30] A. L. Custódio and L. N. Vicente. Using sampling and simplex derivatives in pattern search methods. *SIAM Journal on Optimization*, 18(2):537–555, May 2007.
- [31] J. E. Dennis, Jr., C. J. Price, and I. D. Coope. Direct search methods for nonlinearly constrained optimization using filters and frames. *Optimization and Engineering*, 5(2):123–144, June 2004.
- [32] J. E. Dennis, Jr. and V. Torczon. Direct search methods on parallel machines. *SIAM Journal on Optimization*, 1(4):448–474, November 1991.
- [33] K.R. Fowler, J.P. Reese, C.E. Kees, J.E. Dennis Jr., C.T. Kelley, C.T. Miller, C. Audet, A.J. Booker, G. Couture, R.W. Darwin, M.W. Farthing, D.E. Finkel, J.M. Gablonsky, G. Gray, and T.G. Kolda. Comparison of derivative-free optimization methods for groundwater supply and hydraulic capture community problems. *Advances in Water Resources*, 31(5):743–757, May 2008.

- [34] R. E. Hayes, F. H. Bertrand, C. Audet, and S. T. Kolaczowski. Catalytic combustion kinetics: Using a direct search algorithm to evaluate kinetic parameters from light-off curves. *The Canadian Journal of Chemical Engineering*, 81(6):1192–1199, 2003.
- [35] M. Kokkolaras, C. Audet, and J. E. Dennis, Jr. Mixed variable optimization of the number and composition of heat intercepts in a thermal insulation system. *Optimization and Engineering*, 2(1):5–29, 2001.
- [36] A. L. Marsden, M. Wang, J. E. Dennis, Jr., and P. Moin. Optimal aeroacoustic shape design using the surrogate management framework. *Optimization and Engineering*, 5(2):235–262, 2004.
- [37] A. L. Marsden, M. Wang, J. E. Dennis, Jr., and P. Moin. Suppression of airfoil vortex-shedding noise via derivative-free optimization. *Physics of Fluids*, 16(10):L83–L86, 2004.
- [38] A. L. Marsden, M. Wang, J. E. Dennis, Jr., and P. Moin. Trailing-edge noise reduction using derivative-free optimization and large-eddy simulation. *Journal of Fluid Mechanics*, 572:13–36, February 2007.
- [39] M. S. Ouali, H. Aoudjit, and C. Audet. Optimisation des stratégies de maintenance. *Journal Européen des Systèmes Automatisés*, 37(5):587–605, 2003.
- [40] T. A. Sriver, J. W. Chrissis, and M. A. Abramson. Pattern search ranking and selection algorithms for mixed variable stochastic optimization, 2004. Preprint.