



NOMAD user guide

version 3.5.1

Sébastien Le Digabel and Christophe Tribes

March 22, 2012

HOW TO USE THIS GUIDE:

- NEW USERS OF NOMAD: Section 2 describes how to install the software. Section 3 describes the simplest usage of NOMAD. NOMAD has default values for all of its internal parameters.
- ADVANCED FEATURES OF NOMAD: The more experienced users will find in Section 4 and above ways to tailor the output files and to modify all internal parameters.

PLEASE CITE NOMAD WITH REFERENCES [5, 47].

Contents

1	Introduction	3
2	Installation and examples	4
2.1	Installation procedure	4
2.2	Setting environment variables	7
2.3	Manual compilation of the code	7
2.3.1	X systems: Linux, Unix, and Mac OS X	7
2.3.2	Windows with <code>minGW</code>	7
2.3.3	Windows with <code>Visual C++</code>	8
2.4	Examples	8
2.4.1	Advanced examples	8
2.4.2	Interface examples	9
3	NOMAD batch mode	9
3.1	Creation of a basic parameters file	10
3.2	Basic instructions on blackbox programs	10
4	NOMAD library mode	15
4.1	Definition of the problem	15
4.2	The main function	18
4.2.1	Parameters	18
4.2.2	Evaluator declaration and algorithm run	18
4.2.3	Access to the solution and to optimization data	19
4.3	Other functionalities of the library mode	21
4.3.1	Automatic calls to user-defined functions	21
4.3.2	Create groups of variables	21
4.3.3	Multiple runs	22
5	Parameters description	23
5.1	Parameters describing the problem	23
5.1.1	Basic	23
5.1.2	Advanced	24
5.2	Algorithmic parameters	25
5.2.1	Basic	25
5.2.2	Advanced	26
5.3	Output parameters	28
5.3.1	Basic	28
5.3.2	Advanced	29
5.4	Additional information for some parameters	29
5.4.1	Executable parameters <code>BB_EXE</code> and <code>SGTE_EXE</code>	29
5.4.2	Blackbox input parameter <code>BB_INPUT_TYPE</code>	30
5.4.3	Blackbox output parameter <code>BB_OUTPUT_TYPE</code>	30
5.4.4	Blackbox redirection parameter <code>BB_REDIRECTION</code>	31
5.4.5	Bounds	31
5.4.6	Direction types	31
5.4.7	Output parameters <code>DISPLAY_STATS</code> and <code>STATS_FILE</code>	32
5.4.8	Fixed variables parameter <code>FIXED_VARIABLE</code>	34

5.4.9	Mesh and poll size parameters	34
5.4.10	Opportunistic strategy	34
5.4.11	Scaling parameter SCALING	35
5.4.12	Temporary directory parameter TMP_DIR	35
5.4.13	Group of variable parameter VARIABLE_GROUP	35
5.4.14	Starting point parameter X0	35
6	Special functionalities	36
6.1	Categorical variables	36
6.1.1	Algorithm	36
6.1.2	Mixed variables optimization with NOMAD	37
6.2	Biobjective optimization	38
6.3	Sensitivity analysis	39
6.4	Variable Neighborhood Search (VNS)	40
6.5	Parallel versions	40
6.5.1	The p-Mads method	41
6.5.2	The Coop-MADS method	41
6.5.3	The PSD-MADS method	41
7	Release notes	42
7.1	Version 3.5	42
7.1.1	Minor changes	42
7.2	Previous versions	43
7.2.1	Version 3.4	43
7.2.2	Version 3.3	43
7.2.3	Version 3.2	44
7.2.4	Version 3.1	44
7.3	Future versions	44
	Related publications	48

1 Introduction

NOMAD (Nonsmooth Optimization by Mesh Adaptive Direct Search) is a C++ implementation of the Mesh Adaptive Direct Search (MADS) algorithm [8, 18, 20], designed for constrained optimization of blackbox functions in the form

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

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 . It is also possible to consider a biobjective version of (1): see Section 6.2.

NOMAD is as its best when the functions f and c_j , $j \in J$, are blackbox functions. Such functions are typically the result of expensive computer simulations and have no exploitable property such as derivatives. In addition, these simulations may be contaminated with noise and may fail to give a result even for feasible points.

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.
- Christophe Tribes, GERAD and Département de mathématiques et de génie industriel, École Polytechnique de Montréal.

Version 3.5.3 (and above) of NOMAD is developed by Christophe Tribes. Version 3.0 (and above) of NOMAD is developed by Sébastien Le Digabel. Previous versions were written by Gilles Couture (GERAD).

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 blackbox program that will be called with system calls by NOMAD. In library mode users may define their blackbox 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.

A new user of NOMAD can start to use it easily (see Section 3). NOMAD has default values for all of its internal parameters. The more experienced users will find in this document ways to tailor the output files and to modify the internal parameters. NOMAD should be cited with references [5, 47]. Other relevant papers by the developers are accessible through the NOMAD website 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 and examples

NOMAD is developed in C++ under Linux with the gcc compiler (g++), version 4. The parallel version uses the message passing interface (MPI [55]). In particular, the MPI implementations openMPI, LAM, MPICH, and the Microsoft HPC pack, have been considered.

NOMAD has also been tested on Unix, Mac OS X with Xcode (gcc 4), Windows 7 with minGW (gcc for Windows), and Visual C++ 2010. NOMAD is freely distributed under the GNU Lesser General Public License that can be read in the file `lgpl.txt` provided in the package or at www.gnu.org/licenses.

There are two ways of installing NOMAD: execute the installation program corresponding to your system, or compile the source code. Three files containing the NOMAD package are available on the [website](#). Download the one adapted to your system (Windows, Linux/Unix, or Mac OS X).

2.1 Installation procedure

For Windows, simply execute the downloaded file, and follow the instructions. The NOMAD executable and the NOMAD library included in the Windows package have been constructed with minGW. Please note that, if you are using Visual C++, your programs will not link with this version of the library and you must compile the library yourself.

For **Mac OS X**, open the disk image and copy the **NOMAD** directory into your Applications folder. We suggest that the user chooses an installation directory without space in the name to ease the creation of environment variables. Also choose directories for which you have the adequate writing rights.

For **X** systems, decompress the downloaded zip file where you want to install **NOMAD**, go to the `$NOMAD_HOME/install` directory, and execute the `./install.sh` command.

This script automatically compiles the code and generates the **NOMAD** executable in `$NOMAD_HOME/bin` and the **NOMAD** library in `$NOMAD_HOME/lib`. The script also detects if **MPI** is installed by checking the existence of the command `mpic++`. If so, the parallel **NOMAD** executable and library are generated in the same directories as the scalar version.

Note that this installation procedure may also be applied on **Mac OS X** if the `gcc` compiler is installed (if not, install **Xcode**). If the installation procedure fails, execute directly the provided makefile as described in Section 2.3.1. After installation, you should have the directory structure described by Figure 1.

```

$NOMAD_HOME
|- bin
|- doc
|- examples
|   |- advanced
|   |   |- categorical
|   |   |   |- batch
|   |   |   |- bi_obj
|   |   |   |- single_obj
|   |   |- multi_start
|   |   |- plot
|   |   |- restart
|   |   |- user_search
|   |- basic
|   |   |- batch
|   |   |   |- bi_obj
|   |   |   |- single_obj
|   |   |- library
|   |   |   |- bi_obj
|   |   |   |- single_obj
|   |- interfaces
|   |   |- AMPL
|   |   |- CUTEr
|   |   |- DLL
|   |   |- FORTRAN
|   |   |- GAMS
|   |   |- MATLAB
|   |   |- NOMAD2
|- install
|- lib
|- src
|- tools
|   |- COOP-MADS
|   |- PSD-MADS
|   |- SENSITIVITY

```

Figure 1: Directory structure of the NOMAD package.

2.2 Setting environment variables

The installation programs do not set any environment variables. Defining such variables allows more convenient access to NOMAD. The first variable to be defined should be `$NOMAD_HOME`, whose value is the directory where NOMAD has been installed. This variable is used by the makefiles provided in the examples and is assumed to be defined in this document. Another environment variable to set is the path variable where `$NOMAD_HOME/bin` should be added. This way, you may just type `nomad.exe` to execute NOMAD.

To create your environment variables, on X systems, if your shell is `bash`, add the following lines in the file `.profile` located in your home directory:

```
export NOMAD_HOME=YOUR_NOMAD_DIRECTORY
export PATH=YOUR_NOMAD_DIRECTORY/bin:$PATH
```

In case your shell is `csh` or `tcsh`, add the following lines to the file `.login`:

```
setenv NOMAD_HOME YOUR_NOMAD_DIRECTORY
setenv PATH YOUR_NOMAD_DIRECTORY/bin:$PATH
```

In order for your variables to be active, enter the command `'source .profile'` or `'source .login'`, or simply log out and log in. If you use a different shell, please modify your environment variables accordingly.

On Windows, environment variables are accessible in the **Control Panel|System|Advanced|Environment variables** menu. Please note that environment variables are named differently and `$NOMAD_HOME` corresponds to `%NOMAD_HOME%`. For the parallel version under Windows, you need also to define the `%MPI_HOME%` environment variable corresponding to the home directory of your MPI installation.

2.3 Manual compilation of the code

If the installation program failed, you need to compile the source code located in `$NOMAD_HOME/src` to generate the NOMAD executables. We assume a basic knowledge of makefiles, which are provided for X and Windows systems.

2.3.1 X systems: Linux, Unix, and Mac OS X

Enter the command `'make all'` from a terminal opened in directory `$NOMAD_HOME/src`. This will create the executable file `nomad.exe` located in `$NOMAD_HOME/bin` and the library file `nomad.a` in `$NOMAD_HOME/lib`. For the parallel version, type `'make allmpi'`, after ensuring that the command `mpic++` works. The executable `nomad.MPI.exe` and the library `nomad.MPI.a` should be generated after the compilation. If these `'make'` commands fail, try `'gmake'` instead of `'make'`.

2.3.2 Windows with minGW

Apply the same procedure as in 2.3.1 except that for the parallel version the environment variable `%MPI_HOME%` must be defined depending on your MPI implementation. The executable are `%NOMAD_HOME%\bin\nomad.exe` and `%NOMAD_HOME%\bin\nomad.MPI.exe`, and the libraries `%NOMAD_HOME%\lib\nomad.a` and `%NOMAD_HOME%\lib\nomad.MPI.a`.

2.3.3 Windows with Visual C++

For the scalar version, create a new console and empty project. Choose a name for your project (`'project_name'` for example), 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 to stay consistent with this document. Apply the same procedure to generate the library except that you must create an empty static library project and that you must not insert the file `nomad.cpp` into the project.

For the parallel version you must link your program with MPI. First, you must install a MPI implementation (MPICH or the Microsoft HPC pack, for example). Then, once your project is created, in the project properties, add your MPI library directory to 'Linker Additional Library Directories', and add the MPI library (typically `mpi.lib`) to 'Linker Input Additional Dependencies'. Finally, add the location of the MPI header file to 'Additional Include Directories'.

2.4 Examples

Examples are located in `$NOMAD_HOME/examples`. Some of them use the batch mode described in Section 3 and some of them use the library mode of Section 4. For the library examples, a makefile is included which can be used to generate a scalar executable (command `make`), or a parallel executable (command `make mpi`). The examples are classified into 3 categories: basic examples, advanced examples, and examples illustrating interfaces between NOMAD and various programming or modeling languages.

Basic examples are a good way to start out with NOMAD. The detailed description of the advanced examples is given next.

2.4.1 Advanced examples

- **CATEGORICAL**: Categorical variables on a simple portfolio selection problem. A single-objective and a biobjective version are given. An example with the parameter `NEIGHBORS_EXE` is also provided.
- **MULTI_START**: Multistart program launching multiple instances of MADS. The different starting point are generated following a Latin-Hypercube sample strategy.
- **PLOT**: Illustration of the `NOMAD::Evaluator::update_success()` virtual function allowing to plot information during the NOMAD execution. This example has been developed by Quentin Reynaud.
- **RESTART**: How to make a NOMAD restart and illustration of the user-defined function `NOMAD::Evaluator::update_iteration()`.
- **USER_SEARCH**: How users may code their own search strategy. This example corresponds to a search described in [24]. Other examples on how to design a search strategy can be found in files `$NOMAD_HOME/src/Speculative_Search.*pp`, `LH_Search.*pp`, and `VNS_Search.*pp`. Please note that the MADS theory assumes that trial search points must be lying on the current mesh. Functions `NOMAD::Point::project_to_mesh()` and `NOMAD::Double::project_to_mesh()` are available to perform such projections.

2.4.2 Interface examples

Examples of interfaces included in the NOMAD package are:

- **AMPL**: This interface to the AMPL format uses a library developed by Dominique Orban and available at www.gerad.ca/~orban/LibAmpl/. A `readme.txt` file is given with the example and describes the different steps necessary for the example to work. This example has been written with the help of Dominique Orban and Anthony Guillou.
- **CUTEr**: How to optimize CUTEr [42] test problems.
- **DLL**: Blackbox that is coded inside a dynamic library (a Windows DLL). Single-objective and biobjective versions are available.
- **FORTRAN**: Two examples. First a blackbox problem coded as a FORTRAN routine linked to the NOMAD library. Then a more elaborated example mixing FORTRAN and the NOMAD library where a FORTRAN program is used both to define the problem and to run NOMAD.
- **GAMS**: Optimization on a blackbox that is a GAMS [32] program.
- **MATLAB**: Optimization on a blackbox that is a MATLAB function. In order for this last example to work, the MATLAB MCC compiler must be present, allowing the creation of stand-alone executables from MATLAB functions.
- **NOMAD2**: Program to use NOMAD version 3 on a problem originally designed for the version 2 of the software. This example has been written by Quentin Reynaud.

3 NOMAD batch mode

This section explains how to get started with the NOMAD batch mode and describes all the steps to solve a blackbox problem. The NOMAD batch mode is launched with one argument that corresponds to the name of a parameters text file. The blackbox problem must be coded as a stand-alone program. The different steps are:

1. Install NOMAD following the instructions given in Section 2.
2. Create a directory for your problem. In this document, we use the notation `$PB_DIR` to refer to this directory.
3. Create your problem blackbox, which corresponds to an executable located in `$PB_DIR` (see Section 3.2). This program will output the objective and the constraints.
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.exe`, launch the algorithm with `'$NOMAD_HOME/bin/nomad.exe $PB_DIR/param.txt'`.

At any time, you can type `'nomad.exe -h param_name'` to have information on a specific parameter, as described in Section 7.2.3. Advanced usage of NOMAD is not described in this section: All parameters are described in Section 5 and other examples are given in `$NOMAD_HOME/examples/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.exe $PB_DIR/param.txt`' where `param.txt` is the parameters file (which must be located in the problem directory) and `nomad.exe` is the NOMAD executable.

For basic usage, the following parameters must be defined:

- The number of variables, $n \leq 1000$ (`DIMENSION`).
- The name of the blackbox executable that outputs the objective and the constraints (`BB_EXE`).
- The output types of the blackbox 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 criterion is specified, the algorithm will stop as soon as the mesh size reaches a given tolerance.

An example is given in Figure 2 that corresponds to the parameters file located in `$NOMAD_HOME/examples/basic/batch/single_obj`. Any entry on a line is ignored after the character '#'. The order in which the parameters appear or their case is unimportant.

The two constraints defined in the parameters file in Figure 2 are of different types. 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 Section 5 for the detailed description of all parameters.

3.2 Basic instructions on blackbox programs

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

The problem directory, where the parameters file is located, may have spaces in its name. The blackbox executable may be located in sub-directories of the problem directory, but the names of the sub-directories must be space-free.

The blackbox program returns the evaluation values by displaying them in the standard output. It also returns the value 0 to indicate that the evaluation went well (a simple '`return 0`' instruction in C). Otherwise NOMAD considers that the evaluation failed. The number of values displayed by the blackbox program corresponds to the number of constraints plus one (or two for bi-objective problems) representing the objective function value(s) that one seeks to minimize. The constraints values correspond to constraints of the form $c_j \leq 0$ (for example, the constraint $0 \leq x_1 + x_2 \leq 10$ must 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 blackbox code.

In basic mode, your blackbox program cannot display other data than the objective and constraint values, but the advanced mode allows it to do so. Your code may generate temporary files but it is preferable to include tag numbers to avoid confusion while running a parallel version (see Section 6.5). The advanced parameters described in Section 5.2.2 allow you to include these tags in the blackbox input files. If you already have a blackbox program in a certain format, you need to interface it with a wrapper program to match the NOMAD specifications. If your blackbox program crashes in batch mode, it will not affect NOMAD: The point that caused this crash will simply be tagged as a blackbox failure.

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

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

With `gcc`, you can compile this example with `'g++ -o bb.exe bb.cpp'`, and test it with the text file `x.txt` containing `'0 0 0 0 0'`, by entering the command `'bb.exe x.txt'`. This test 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 ($c_2(x) = 20 > 0$).

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 g1 <= 0 and
		# g2 <= 0; 'PB' and 'EB'
		# correspond to constraints that
		# are treated by the Progressive
		# and 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 <= 5, x_2 <= 6, x_3 <= 7
		# x_4 and x_5 have no bounds
MAX_BB_EVAL	100	# the algorithm terminates when
		# 100 blackbox evaluations have
		# been made
TMP_DIR	/tmp	# indicates a directory 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 Section 5 or with the command 'nomad.exe -h param_name'.

```

#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 blackbox program. This code corresponds to the file `bb.cpp` in `$NOMAD_HOME/examples/basic/batch/single_obj`.

```

NOMAD - version 3.5.3 - www.gerad.ca/nomad

Copyright (C) 2001-2011 {
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
Christophe Tribes     - 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

      3 0.0000000000
     12 -1.0000000000
     14 -3.0000000000
     99 -3.5000000000
    100 -3.5000000000

} end of run (max number of blackbox evaluations)

blackbox evaluations   : 100
best infeasible solution: ( 1.1 1.2 1.3 1 -4 ) h=0.14 f=-4
best feasible solution  : ( 2.75 0.6 0.65 0.5 -3.5 ) h=0 f=-3.5

```

Figure 4: Output given by NOMAD on the blackbox problem coded in Figure 3 with parameters file in Figure 2.

NOMAD is flexible enough so that blackbox codes can be coded differently and with more sophistication in the advanced mode (see Section 4).

Figure 4 shows the display that the execution of NOMAD produces for the blackbox program in Figure 3 with the parameters file in Figure 2. Notice that the first feasible point has been found after 3 blackbox evaluations. In this case, the starting point $x = (0 \ 0 \ 0 \ 0)^T$ violates the second constraint, which is treated by the extreme barrier approach. In such a situation, NOMAD launches an initial optimization, called the *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.

4 NOMAD library mode

This section explains how to create a C++ program able to call the NOMAD routines using the pre-compiled NOMAD static library. It is supposed that the library is correctly installed and that the environment variable `$NOMAD_HOME` is defined. If not, you must specify the installation directory of NOMAD in the makefile. Explanations are given for Linux and g++ and for the scalar library, but are similar for Windows and minGW or Visual C++, and for the parallel library. 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 `NOMAD::Point`. A lot of functionalities have been coded for these classes, which are visible in files `Double.hpp` and `Point.hpp`. All NOMAD class files are named like the classes and are located in the 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 namespace `NOMAD` is used for all NOMAD types, and you must type `NOMAD::` in front of all types unless you type `'using namespace NOMAD;'` at the beginning of your program.

The example shown in this section corresponds to files located in the directory `$NOMAD_HOME/examples/basic/library/single.obj`. It is identical to the example shown in Section 3, except that no temporary files are used, and no system calls are made. For this example, just one C++ source file is used, but there could be a lot more. Other examples can be found in `$NOMAD_HOME/examples` and in the main function of NOMAD located in the file `$NOMAD_HOME/src/nomad.cpp` which implements the NOMAD batch mode. This illustrates the fact that even in library mode a parameters file may be used and system calls performed.

As a first task, a makefile needs 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. Such makefiles are given at various places inside the `examples` directory.

We now describe the other steps required for the creation of the source file `basic_lib.cpp`, which includes the header file `nomad.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 `basic_lib.exe` is created and can be executed.

4.1 Definition of the problem

Describing the blackbox problem directly in 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 `NOMAD::Evaluator` in single-objective optimization and from `NOMAD::Multi_Obj_Evaluator` in multi-objective mode (see header files `Evaluator.hpp` and `Multi_Obj_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 Figure 6. Note that const and non-const versions of the method are available.

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 result of the evaluation. The coordinates are accessed with the operator `[]` (`x[0]` for the first coordinate) and outputs are set with the method `NOMAD::Eval_Point::set_bb_output()` (`x.set_bb_output(0,v)` to set the objective function value to `v` if the objective has been defined at the first position). Constraints must 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

```

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

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

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

```

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

```

Figure 6: Prototype of method `NOMAD::Evaluator::eval_x()`. A non-const version is also available.

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 may be used to interrupt an expensive evaluation if the constraint violation value h grows larger than h_{max} . See [20] 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 blackbox evaluation, and `false` otherwise (for example, if the user interrupts an evaluation with the h_{max} criterion before it costs some expensive computations, then set `count_eval` to `false`).

If a surrogate function is to be used, then its evaluation routine should be coded in the method `eval_x()`. First, to indicate that a surrogate can be computed, the user must set the parameter `HAS_SGTE` to `yes`, via the method `NOMAD::Parameters::set_HAS_SGTE()`. Then, in `eval_x()`, the test `'if (x.get_eval_type()==SGTE)'` must be made to differentiate an evaluation with the true function f or with the surrogate. More notes on surrogates are given in Section 5.1.2.

Another possibility for the designer of `eval_x()` is the ability to define a priority for the trial point x via the method `NOMAD::Eval_Point::set_user_eval_priority()`. Points with higher priorities will be evaluated first. For more details, see the code in `Priority_Eval_Point.*pp`.

The function `eval_x()` should return `true` if the evaluation succeeded, and `false` if the evaluation failed.

Finally, note that the call to `eval_x()` inside the NOMAD code is inserted into a `try` block. This

means that if an error is detected inside the `eval_x()` function, an exception should be thrown. The choice for the type of this exception is left to the user, but `NOMAD::Exception` is available (see `Exception.*pp`). If an exception is thrown by the user-defined function, then the associated evaluation is tagged as a failure and not counted unless the user explicitly set the flag `count_eval` to `true`. Additionally, the user-defined function can test on whether CTRL-C has been pressed by using the method `NOMAD::Evaluator::get_force_quit()`. This allows managing the termination of a costly black-box evaluation within `eval_x`.

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

    ~My_Evaluator ( void ) {}

    bool eval_x ( NOMAD::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 blackbox evaluation
        return true;       // the evaluation succeeded
    }
};
```

Figure 7: Example of a user class defining a hard-coded blackbox problem.

Of course, more elaborated `NOMAD::Evaluator` subclasses may be designed in order to consider some additional problem-related parameters. Such an example can be found in the source files `Multi_Obj_Evaluator.*pp` where some weights are defined to change the objective function of the problem between successive optimizations (this example correspond to the BiMADS algorithm [26]).

The virtual method `NOMAD::Evaluator::update_success()` can also be subclassed. The corresponding derived method will be automatically invoked every time a new improvement is made. Note that the automatic calls to this method can be enabled/disabled with `NOMAD::Evaluator_Control::set_call_user_update_success()`.

Another virtual method defined in the class `NOMAD::Evaluator` is `compute_f()`. This method allows the user to compute the value of the objective function directly from the blackbox outputs. This is used by the BiMADS algorithm. If `compute_f()` is not user-defined, then NOMAD simply takes the value of f as the first OBJ output from the blackbox.

4.2 The main function

Once your problem has been defined, the main function can be written. NOMAD routines may throw C++ exceptions, so it is recommended that you put your code into a `try` block. In addition, functions `NOMAD::begin()` and `NOMAD::end()` must be called at the beginning and at the end of the main function. `NOMAD::Slave::stop_slaves()` has also to be called at the end of the main function if parallelism is used.

4.2.1 Parameters

First, a `NOMAD::Parameters` object needs to be declared. Parameters are defined similarly as in batch mode and each parameter `PNAME` is set with the method `NOMAD::Parameters::set_PNAME()`. In order to see all the options, use the help '`nomad.exe -h param_name`', or refer to the detailed list of parameters in Section 5, or to the header file `Parameters.hpp`. NOMAD additional C++ types necessary for the calls to `NOMAD::Parameters` functions can be found in the file `defines.hpp`. An example is given in Figure 8. This example is taken from file `basic_lib.cpp` located in `$NOMAD_HOME/examples/basic/library/single_obj` and corresponds to the same parameters as given in Figure 2 except for `BB_EXE` which is not required.

In library mode it is possible to provide the parameters programmatically or by reading from a file, with `NOMAD::Parameters::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 path instead of the default location `./`. To display the parameters described by a `NOMAD::Parameters` object `p`, use the instruction `'cout << p << endl;'`.

Once that all parameters are set, the method `NOMAD::Parameters::check()` must be invoked to validate the parameters. The algorithm will not run with a non-checked `NOMAD::Parameters` object. It is not even possible to access data from an object of this class while not checked. If parameters are changed, `check()` must be invoked again before a new run can be conducted. Notice that the call to `check()` may be bypassed by using `NOMAD::Parameters::force_check_flag()` but only advanced users should use it.

4.2.2 Evaluator declaration and algorithm run

The MADS algorithm is implemented in the `NOMAD::Mads` class. Objects of this class are created with a `NOMAD::Parameters` object and an `NOMAD::Evaluator` object as arguments. In the example described here, the `NOMAD::Evaluator` object corresponds to an object of type `My_Evaluator`. A `NULL` pointer may also be used instead of the `NOMAD::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 blackbox program and system calls. When an `NOMAD::Evaluator` object is used, parameters `BB_EXE` and `SGTE_EXE` are ignored. A more advanced `NOMAD::Mads` constructor with user-created caches is also available in `$NOMAD_HOME/src/Mads.hpp`.

Once the `NOMAD::Mads` object is declared, run the algorithm with `NOMAD::Mads::run()` (or `NOMAD::Mads::multi_run()` for multi-objective optimization). An example is shown in Figure 9.

It is also possible for the user to redefine the virtual method `NOMAD::Evaluator::list_of_points_preprocessing()` to indicate a preprocessing strategy that will be applied by the algorithm before each series of evaluations is made. All evaluation points may then be modified according to this strategy. See `$NOMAD_HOME/src/Evaluator.hpp` for the header of this method.

4.2.3 Access to the solution and to optimization data

In the example of `$NOMAD_HOME/examples/basic/library/single_obj`, final information is displayed via a call to the operator `<<` at the end of `NOMAD::Mads::run()`. More specialized access to solution and optimization data is allowed. To access the best feasible and infeasible points, use the methods `NOMAD::Mads::get_best_feasible()` and `NOMAD::Mads::get_best_infeasible()`. To access optimization data or statistics, call the method `NOMAD::Mads::get_stats()` which returns access to a `NOMAD::Stats` object. Then, use the access methods defined in `Stats.hpp`. For example, to display the number of blackbox evaluations, write:

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

```

// display:
NOMAD::Display out ( std::cout );

// parameters creation:
NOMAD::Parameters p ( out );

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

// definition of output types:
vector<NOMAD::bb_output_type> bbot (3);
bbot[0] = NOMAD::OBJ;
bbot[1] = NOMAD::PB;
bbot[2] = NOMAD::EB;
p.set_BB_OUTPUT_TYPE ( bbot );

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

// lower bounds: all var. >= -6:
p.set_LOWER_BOUND ( NOMAD::Point ( 5 , -6.0 ) );

// upper bounds (x_4 and x_5 have no upper bounds):
NOMAD::Point ub ( 5 );
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

// parameters validation:
p.check();

```

Figure 8: Example of parameters creation in library mode.

```

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

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

```

Figure 9: Evaluator and Mads objects usage.

4.3 Other functionalities of the library mode

4.3.1 Automatic calls to user-defined functions

Virtual methods are automatically invoked by NOMAD at some special events of the algorithm. These methods are left empty by default and you may redefine them so that your own code is automatically called. These virtual methods are defined in the `NOMAD::Evaluator` and `NOMAD::Multi_Obj_Evaluator` classes and are detailed below:

- `NOMAD::Evaluator::list_of_points_preprocessing()`: Called before the evaluation of a list of points (it allows the user to pre-process the points to be evaluated).
- `NOMAD::Evaluator::update_iteration()`: Invoked every time a MADS iteration is terminated.
- `NOMAD::Evaluator::update_success()`: Invoked when a new incumbent is found (single-objective) or when a new Pareto point is found (biobjective).
- `NOMAD::Multi_Obj_Evaluator::update_mads_run()`: For biobjective problems, this method is called every time a single MADS run is terminated.

It is possible to disable the automatic calls to these methods, with the functions `NOMAD::Mads::enable_user_calls()` and `NOMAD::Mads::disable_user_calls()`, or with the parameters `USER_CALLS_ENABLED` and `EXTENDED_POLL_ENABLED`. These parameters are automatically set to `yes`, except during the extended poll for categorical variables and during the VNS search.

4.3.2 Create groups of variables

This section gives some explanations about creating groups of variables in library mode. See Section 5.4.13 for defining such groups in batch mode.

Groups of variable are created with the method `NOMAD::Parameters::set_VARIABLE_GROUP()` which has two different prototypes. The method must be called each time a new group is created. For both versions of the function, the user needs to the set of indices of the variables composing each group.

In NOMAD, a group of variable generates its own polling directions. The most complete prototype of `set_VARIABLE_GROUP()` allows to choose the types of these directions, for the primary and secondary polls. The detailed types of directions can be found in file `defines.hpp` and the `enum` type `direction_type`. The simplified prototype uses ORTHOMADS types of directions by default. In all cases a Halton seed must be provided, which is not considered if direction types do not correspond to ORTHOMADS. Otherwise, a value must be provided. This value should be larger than the n th prime number, and ideally be different for each group of variables. The method `NOMAD::Directions::get_max_halton_seed()` is available in order to get the highest Halton seed that has been used, and help determine such a value. It is also possible to use the method `NOMAD::Directions::compute_halton_seed()` which directly computes the Halton seed as the n th prime number.

Finally the function `NOMAD::Parameters::reset_variable_groups()` may be called to reset the groups of variables. Remember also that after a modification to a `Parameters` object is made, the method `NOMAD::Parameters::check()` needs to be called.

4.3.3 Multiple runs

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

A simple solution for doing that is to declare the `NOMAD::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 deleted with `NOMAD::Mads` objects).

```
{
    NOMAD::Mads mads ( p , &ev , cout );

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

// some changes...

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

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

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

Another solution consists in using the `NOMAD::Mads::reset()` method between consecutive runs and to keep the `NOMAD::Mads` object in a more global scope. The method takes two boolean arguments (set to `false` by default), `keep_barriers` and `keep_stats`, indicating if the barriers (true and surrogate) and statistics must be reseted between the two runs. An example is shown in Figure 11.

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

// some changes...
mads.reset();

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

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

Two examples showing multiple MADS runs are described in the advanced examples directory.

5 Parameters description

This section describes the parameters for the optimization problem definition, the algorithmic parameters and parameters to manage output information. Additional information can be obtained by executing the command `'nomad.exe -h'`, to see all parameters, or `'nomad.exe -h PARAM_NAME'` for a particular parameter.

In library mode, parameters are defined via a `NOMAD::Parameters` object and methods `NOMAD::Parameters::set_PARAM_NAME()`, where `PARAM_NAME` is the name used in this section. It is also possible to read a parameters file in library mode, with the method `NOMAD::Parameters::read()`.

In batch mode, the problem directory is automatically determined by NOMAD. It can be defined in library mode with `NOMAD::Parameters::set_PROBLEM_DIR()`.

All the entries of a line are ignored after the character `'#'`. Except for the file names, all strings and parameter names are case insensitive (`'DIMENSION 2'` is the same as `'Dimension 2'`). File names refer to files in the problem directory. To indicate a file name containing spaces, use quotes (`"name"` or `'name'`). These names may include directory information relatively to the problem directory. The problem directory will be added to the names, unless the `'$'` character is used in front of the names. For example, if a blackbox executable is run by the command `'python script.py'`, define parameter `BB_EXE` with argument `'$python script.py'`.

Some parameters consists of a list of variable indices taken from 0 to $n-1$. Variable indices may be entered individually or as a range with format `'i-j'`. Character `'*'` may be used to replace `'0-n-1'` (where n is the number of variables). Other parameters require arguments of type `boolean`: these values may be entered with the strings `yes`, `no`, `y`, `n`, `0`, or `1`. Finally, some parameters need vectors as arguments, use `(v1 v2 ... vn)` for those. Characters `'-'`, `'inf'`, `'-inf'` or `'+inf'` are accepted to enter undefined real values (NOMAD considers $\pm\infty$ as an undefined value).

The following subsections show tables describing all NOMAD parameters. Parameters are classified into problem, algorithmic and output parameters. For each of these classes, basic and advanced parameters are described separately.

5.1 Parameters describing the problem

5.1.1 Basic

name	arguments	description	default
<code>BB_EXE</code>	list of strings; see 5.4.1	blackbox executables (required in batch mode)	none
<code>BB_INPUT_TYPE</code>	see 5.4.2	blackbox input types	<code>* R</code> (all real)
<code>BB_OUTPUT_TYPE</code>	see 5.4.3	blackbox output types (required)	none
<code>DIMENSION</code>	integer	n the number of variables (required, $n \leq 1000$)	none
<code>LOWER_BOUND</code>	see 5.4.5	lower bounds	none
<code>UPPER_BOUND</code>	see 5.4.5	upper bounds	none

5.1.2 Advanced

name	arguments	description	default
FIXED_VARIABLE	see 5.4.8	fixed variables	none
PERIODIC_VARIABLE	index range	define variables in the range to be periodic (bounds required)	none
SGTE_COST	integer c	the cost of c surrogate evaluations is equivalent to the cost of one blackbox evaluation	∞
SGTE_EVAL_SORT	bool	if surrogates are used to sort list of trial points	yes
SGTE_EXE	list of strings; see 5.4.1	surrogate executables	none
VARIABLE_GROUP	index range	defines a group of variables; see 5.4.13	none

Surrogates, or surrogate functions, are cheaper blackbox functions that are used, at least partially, instead of the true function f to minimize. The current version of NOMAD uses only static surrogates which are not updated during the algorithm and which are provided by the user. If such functions are defined, NOMAD will use them to drive its search. See [\[31\]](#) for a survey on surrogate optimization.

5.2 Algorithmic parameters

5.2.1 Basic

name	arguments	description	default
DIRECTION_TYPE	see 5.4.6	type of directions for the poll	ORTHO
F_TARGET	reals, f or (f1 f2)	NOMAD terminates if $f_i(x_k) \leq \mathbf{f_i}$ for all objective functions	none
HALTON_SEED	integer	Halton seed for ORTHO-MADS [8]	n th prime number
INITIAL_MESH_SIZE	see 5.4.9	Δ_0^m [18]	r0.1 or based on X0
LH_SEARCH	2 integers: p0 and pi	LH (Latin-Hypercube) search (p0: initial, pi: iterative); see 6.2 for biobjective	none
MAX_BB_EVAL	integer	maximum number of blackbox evaluations; see 6.2 for biobjective	none
MAX_TIME	integer	maximum wall-clock time (in seconds)	none
MODEL_EVAL_SORT	bool	enable or not the ordering of trial points based on a quadratic model	yes
MODEL_SEARCH	bool	enable or not the search strategy using quadratic models	yes
MULTI_NB_MADS_RUNS	integer	number of MADS runs	see 6.2
MULTI_OVERALL_BB_EVAL	integer	max number of blackbox evaluations for all MADS runs	see 6.2
OPPORTUNISTIC_EVAL	bool	opportunistic strategy; see 5.4.10	yes
OPPORTUNISTIC_LH	bool	opportunistic strategy for LH search; see 6.2 for biobjective	see 5.4.10
SEED	integer or NONE	random seed; NONE or a negative integer to define a seed that will be different at each run	NONE
TMP_DIR	string	temporary directory for blackbox i/o files; see 5.4.12	problem directory
VNS_SEARCH	bool or real	VNS search; see 6.4	no
X0	see 5.4.14	starting point(s)	best point from a cache file or from an initial LH search

5.2.2 Advanced

name	arguments	description	default
ASYNCHRONOUS	bool	asynchronous strategy for the parallel version; see 6.5	yes
BB_INPUT_INCLUDE_SEED	bool	if the random seed is put as the first entry in blackbox input files	no
BB_INPUT_INCLUDE_TAG	bool	if the tag of a point is put as an entry in blackbox input files	no
BB_REDIRECTION	bool	if NOMAD manages the creation of blackbox output files; see 5.4.4	yes
CACHE_SEARCH	bool	enable or disable the cache search (useful with extern caches)	no
EPSILON	real	precision on reals	1E-13
EXTENDED_POLL_ENABLED	bool	if no, the extended poll for categorical variables is disabled	yes
EXTENDED_POLL_TRIGGER	real	trigger for categorical variables; value may be relative; see 6.1	r0.1
H_MAX_0	real	initial value of h_{max} (will be eventually decreased throughout the algorithm)	1E+20
H_MIN	real v	x is feasible if $h(x) \geq v$	0.0
H_NORM	norm type in {L1, L2, Linf}	norm used to compute h	L2
HAS_SGTE	bool	indicates if the problem has a surrogate (only necessary in library mode)	no or yes if SGTE_EXE is defined
INITIAL_MESH_INDEX	integer	initial mesh index ℓ_0 [8]	0
L_CURVE_TARGET	real	NOMAD terminates if it detects that the objective may not reach this value	none
MAX_CACHE_MEMORY	integer	NOMAD terminates if the cache reaches this memory limit expressed in MB	2000
MAX_CONSECUTIVE_FAILED_ITERATIONS	integer	max number of MADS failed iterations	none
MAX_EVAL	integer	max number of evaluations (includes cache hits and blackbox evaluations, does not include surrogate eval)	none
MAX_ITERATIONS	integer	max number of MADS iterations	none
MAX_MESH_INDEX	integer	max mesh index ℓ_{max} [8]	none
MAX_SGTE_EVAL	integer	max number of surrogate evaluations	none
MAX_SIM_BB_EVAL	integer	max number of simulated blackbox evaluations (includes initial cache hits)	none

name	arguments	description	default
MESH_COARSENING_EXPONENT	integer	w^+ [18]	1
MESH_REFINING_EXPONENT	integer	w^- [18]	-1
MESH_UPDATE_BASIS	real	τ [18]	4.0
MIN_MESH_SIZE	see 5.4.9	Δ_{min}^m [18]	none
MIN_POLL_SIZE	see 5.4.9	Δ_{min}^p [18]	none or 1 for int/bin variables
MODEL_EVAL_SORT_CAUTIOUS	bool	if the model ordering strategy is cautious	yes
MODEL_SEARCH_MAX_TRIAL_PTS	integer	limit on the number of trial points for one model search	4
MODEL_SEARCH_OPTIMISTIC	bool	if model search is optimistic or not	yes
MODEL_SEARCH_PROJ_TO_MESH	bool	if model search trial points are projected to the mesh	yes
MODEL_QUAD_MAX_Y_SIZE	integer	sup. limit on the size of in- terpolation sets for quadratic models	500
MODEL_QUAD_MIN_Y_SIZE	integer or string	inf. limit on the size of in- terpolation sets for quadratic models	N+1
MODEL_QUAD_RADIUS_FACTOR	real	quadratic model search radius factor	2.0
MODEL_QUAD_USE_WP	bool	enable the strategy to main- tain well-poisedness with quadratic models	no
MULTI_F_BOUNDS	4 reals	see 6.2	none
MULTI_FORMULATION	string	see 6.2	PRODUCT or DIST.L2
MULTI_USE_DELTA_CRIT	bool	see 6.2	no
NEIGHBORS_EXE	string	neighborhood executable for categorical variables in batch mode	none
OPPORTUNISTIC_CACHE_SEARCH	bool	opportunistic strategy for cache search	no
OPPORTUNISTIC_LUCKY_EVAL	bool	see 5.4.10	none
OPPORTUNISTIC_MIN_EVAL	integer	see 5.4.10	none
OPPORTUNISTIC_MIN_F_IMPRVMT	real	see 5.4.10	none
OPPORTUNISTIC_MIN_NB_SUCCESS	integer	see 5.4.10	none
OPT_ONLY_SGTE	bool	minimize only with surrogates	no
RHO	real	ρ parameter of the progressive barrier	0.1
SCALING	see 5.4.11	scaling on the variables	none
SEC_POLL_DIR_TYPE	see 5.4.6	type of directions for the sec- ondary poll	see 5.4.6
SNAP_TO_BOUNDS	bool	snap to boundary trial points that are generated outside bounds	yes
SPECULATIVE_SEARCH	bool	MADS speculative search [18]	yes
STAT_SUM_TARGET	real	NOMAD terminates if STAT_SUM reaches this value	none
STOP_IF_FEASIBLE	bool	NOMAD terminates if it gener- ates a feasible solution	no
USER_CALLS_ENABLED	bool	if no, the automatic calls to user functions are disabled	yes

5.3 Output parameters

Three different level of display can be set via the parameter `DISPLAY_DEGREE`, and these levels may be set differently for 4 different sections of the algorithm (general displays, search and poll displays and displays for each iteration data). The three different levels can be entered with an integer in $[0; 2]$, but also with the strings `'NO_DISPLAY'`, `'NORMAL_DISPLAY'`, or `'FULL_DISPLAY'`. If the maximum level of display is set, then the algorithm informations are displayed within indented blocks. These blocks ease the interpretation of the algorithm logs when read from a text editor. The characters used to mark the beginning and the end of these blocks can be changed with the parameters `OPEN_BRACE` and `CLOSED_BRACE`.

From the implementation point of view, constants have been introduced for the display degrees: `NOMAD::NO_DISPLAY`, `NOMAD::NORMAL_DISPLAY`, and `NOMAD::FULL_DISPLAY`. In addition, a specific class is now responsible for all program displays. This class is named `NOMAD::Display` and is constructed from a `std::ostream` object such as `std::cout`. The use of a `NOMAD::Display` object is similar to the use of a `std::ostream` object, except that the outputs are organized within indented blocks. The method `NOMAD::Parameters::out()` provides access to the `NOMAD::Display` object used by the program.

5.3.1 Basic

name	arguments	description	default
<code>CACHE_FILE</code>	string	cache file; if the file does not exist, it will be created	none
<code>DISPLAY_ALL_EVAL</code>	bool	if yes all points are displayed with <code>DISPLAY_STATS</code> and <code>STATS_FILE</code>	no
<code>DISPLAY_DEGREE</code>	integer in $[0; 2]$ or a string with four digits; see 5.3.2	0: no display; 2: full display	1
<code>DISPLAY_STATS</code>	list of strings	what informations is displayed at each success; see 5.4.7	see 5.4.7
<code>HISTORY_FILE</code>	string	file containing all trial points with format (<code>x1 x2 ... xn</code>) on each line; includes multiple evaluations	none
<code>SOLUTION_FILE</code>	string	file to save the current best feasible point	none
<code>STATS_FILE</code>	a string <code>file_name</code> plus a list of strings	the same as <code>DISPLAY_STATS</code> but for a display into file <code>file_name</code>	none

5.3.2 Advanced

name	arguments	description	default
ADD_SEED_TO_FILE_NAMES	bool	if the seed is added to the file names corresponding to parameters HISTORY_FILE, SOLUTION_FILE and STATS_FILE	yes
CACHE_SAVE_PERIOD	integer i	the cache files are saved every i iterations (disabled for biobjective)	25
CLOSED_BRACE	string	displayed at the end of indented blocks	'{'
DISPLAY_DEGREE	string with four digits, each in [0; 2]	1st digit: general display; 2nd digit: search display; 3rd digit: poll display; 4th digit: iterative display; example: DISPLAY_DEGREE 0010	1111
INF_STR	string	used to display infinity	'inf'
OPEN_BRACE	string	displayed at the beginning of indented blocks	'}'
POINT_DISPLAY_LIMIT	integer	maximum number of point coordinates that will be displayed at screen (-1 for no limit)	20
SGTE_CACHE_FILE	string	surrogate cache file (can not be the same as CACHE_FILE)	none
UNDEF_STR	string	used to display undefined values	'-'

5.4 Additional information for some parameters

5.4.1 Executable parameters BB.EXE and SGTE.EXE

In batch mode, BB.EXE indicates the names of the blackboxes executables. In library mode, it is optional as a custom `NOMAD::Evaluator` class may be written with its own `eval_x()` method. A single string may be given if a single blackbox is used and gives several outputs. It is also possible to indicate several blackbox executables. If the character '\$' is put at first position of a string, this string is considered as a global command or file and no path is added. We give the following examples:

```

BB_EXE          bb.exe          # defines that 'bb.exe' is an
BB_OUTPUT_TYPE OBJ EB EB        # executable with 3 outputs

BB_EXE          bb1.exe bb2.exe  # defines two blackboxes
BB_OUTPUT_TYPE OBJ      EB        # 'bb1.exe' and 'bb2.exe'
                                   # with one output each

BB_EXE "dir $with $spaces/bb.exe # use '$' to describe a
                                   # path with spaces

BB_EXE "$python bb.py"          # the blackbox is a python
                                   # script: it is run with
                                   # command
                                   # 'python PROBLEM_DIR/bb.py'

BB_EXE "$nice bb.exe"           # to run PROBLEM_DIR/bb.exe
                                   # in nice mode on X systems

```

The parameter `SGTE_EXE` associates surrogate executables with blackbox executables. It may be entered with two formats: `'SGTE_EXE bb_exe sgte_exe'` to associate executables `bb_exe` and `sgte_exe`, or `'SGTE_EXE sgte_exe'` when only one blackbox executable is used. Surrogates must display the same number of outputs as their associated blackboxes.

5.4.2 Blackbox input parameter `BB_INPUT_TYPE`

This parameter indicates the types of each variable. It may be defined once with a list of n input types with format `(t1 t2 ... tn)` or several times with index ranges and input types. Input types are values in $\{R, C, B, I\}$ or $\{Real, Cat, Bin, Int\}$. `R` is for real/continuous variables, `C` for categorical variable, `B` for binary variables, and `I` for integer variables. The default type is `R`.

5.4.3 Blackbox output parameter `BB_OUTPUT_TYPE`

This parameter defines the types of the values that the blackbox displays. The arguments are a list of m types, where m is the number of outputs of the blackbox. At least one of these values must correspond to the objective function that NOMAD minimizes. If two outputs are tagged as objectives, then the BiMADS algorithm will be executed. Other values typically are constraints of the form $c_j(x) \leq 0$, and the blackbox must display the left hand side of the constraint with this format. A certain terminology is used to describe the different types of constraints. This terminology can be consulted in [20]. `EB` constraints correspond to constraints that need to be always satisfied (*unrelaxable constraints*). The technique used to deal with those is the *extreme barrier* approach, consisting in simply rejecting the infeasible points. `PB`, `PEB`, and `F` constraints correspond to constraints that need to be satisfied only at the solution, and not necessarily at intermediate points (*relaxable constraints*). More precisely, `F` constraints are treated with the *filter* approach [17], and `PB` constraints are treated with the *progressive barrier* approach [20]. `PEB` constraints are treated first with the progressive barrier, and once satisfied, with the extreme barrier [22]. There may be another type of constraints, the *hidden constraints*, but these only appear inside the blackbox during an execution, and thus they cannot be indicated in advance to NOMAD (when such a constraint is violated, the evaluation simply fails and the point is not considered). If the user is not sure about the nature of its constraints, we suggest using the keyword `CSTR`, which correspond by default to `PB` constraints.

There may be other types of outputs. All the types are:

<code>CNT_EVAL</code>	Must be 0 or 1: count or not the blackbox evaluation (equivalent to the argument <code>cnt_eval</code> of <code>NOMAD::Evaluator::eval_x()</code>).
<code>EB</code>	Constraint treated with Extreme Barrier (infeasible points are ignored).
<code>F</code>	Constraint treated with filter approach [17].
<code>NOTHING</code> or <code>-</code>	The output is ignored.
<code>OBJ</code>	Objective value to minimize.
<code>PB</code> or <code>CSTR</code>	Constraint treated with Progressive Barrier [20].
<code>PEB</code>	Hybrid constraint <code>PB/EB</code> [22].
<code>STAT_AVG</code>	Average of this value will be computed for all blackbox calls (must be unique).
<code>STAT_SUM</code>	Sum of this value will be computed for all blackbox calls (must be unique).

Please note that F constraints are not compatible with CSTR, PB or PEB. However, EB can be combined with F, CSTR, PB or PEB.

5.4.4 Blackbox redirection parameter BB_REDIRECTION

If this parameter is set to **yes** (default), NOMAD manages the creation of the blackbox output file when the blackbox is executed via a system call (the redirection ‘>’ is added to the system command). If no, then the blackbox must manage the creation of its output file named `TMP_DIR/nomad.SEED.TAG.output`. Values of `SEED` and `TAG` can be obtained in the blackbox input files created by NOMAD and given as first argument of the blackbox, only if parameters `BB_INPUT_INCLUDE_SEED` and `BB_INPUT_INCLUDE_TAG` are both set to **yes**. `TMP_DIR` is specified by the user. If no, `TMP_DIR` is the problem directory.

5.4.5 Bounds

Parameters `LOWER_BOUND` and `UPPER_BOUND` are used to define bounds on variables, and take similar arguments as parameter `FIXED_VARIABLE` (see 5.4.8). For example, with $n = 7$,

<code>LOWER_BOUND</code>	0-2	-5.0
<code>LOWER_BOUND</code>	3	0.0
<code>LOWER_BOUND</code>	5-6	-4.0
<code>UPPER_BOUND</code>	0-5	8.0

is equivalent to

```

LOWER_BOUND ( -5 -5 -5 0 - -4 -4 ) # '-' or '-inf' means that x_4
                                     # has no lower bound
UPPER_BOUND ( 8 8 8 8 8 8 inf )    # '-' or 'inf' or '+inf' means
                                     # that x_6 has no upper bound.

```

These two sequences define the following bounds

$$\left\{ \begin{array}{lll} -5 \leq & x_1 & \leq 8 \\ -5 \leq & x_2 & \leq 8 \\ -5 \leq & x_3 & \leq 8 \\ 0 \leq & x_4 & \leq 8 \\ & x_5 & \leq 8 \\ -4 \leq & x_6 & \leq 8 \\ -4 \leq & x_7 & . \end{array} \right.$$

5.4.6 Direction types

The types of direction correspond to the arguments of parameters `DIRECTION_TYPE` and `SEC_POLL_DIR_TYPE`. Up to 4 strings may be employed to describe one direction type. These 4 strings are `s1` in $\{\text{ORTHO}, \text{LT}, \text{GPS}\}$, `s2` in $\{\emptyset, 1, 2, N+1, 2N\}$, `s3` in $\{\emptyset, \text{STATIC}, \text{RANDOM}\}$, and `s4` in $\{\emptyset, \text{UNIFORM}\}$. If only 1, 2 or 3 strings are given, defaults are considered for the others. Combination of these strings may describe the following 14 direction types:

	s1	s2	s3	s4	direction types
1	ORTHO	1			ORTHOMADS, 1.
2	ORTHO	2			ORTHOMADS, 2.
3	ORTHO				ORTHOMADS, 2n.
3	ORTHO	2N			ORTHOMADS, 2n.
4	LT	1			LT-MADS, 1.
5	LT	2			LT-MADS, 2.
6	LT	N+1			LT-MADS, n+1.
7	LT				LT-MADS, 2n.
7	LT	2N			LT-MADS, 2n.
8	GPS	BIN			GPS for binary variables.
9	GPS	N+1			GPS, n+1, static.
9	GPS	N+1	STATIC		GPS, n+1, static.
10	GPS	N+1	STATIC	UNIFORM	GPS, n+1, static, uniform angles.
11	GPS	N+1	RAND		GPS, n+1, random.
12	GPS	N+1	RAND	UNIFORM	GPS, n+1, random, uniform angles.
13	GPS				GPS, 2n, static.
13	GPS	2N			GPS, 2n, static.
13	GPS	2N	STATIC		GPS, 2n, static.
14	GPS	2N	RAND		GPS, 2n, random.

GPS directions correspond to the coordinate directions. LT and ORTHO directions correspond to the implementations LT-MADS [18] and ORTHOMADS [8] of MADS. The integer indicated after GPS, LT and ORTHO corresponds to the number of directions that are generated at each poll. The 14 different direction types may be chosen together by specifying `DIRECTION_TYPE` or `SEC_POLL_DIR_TYPE` several times. If nothing indicated, ORTHO is considered for the primary poll, and default direction types for the secondary poll are ORTHO 1 or 2, LT 1 or 2, and GPS N+1 STATIC depending on the value of `DIRECTION_TYPE`.

5.4.7 Output parameters `DISPLAY_STATS` and `STATS_FILE`

These parameters display information each time a new feasible incumbent is found. `DISPLAY_STATS` displays at the standard output and `STATS_FILE` writes a file. These parameters need a list of strings as argument, **without any quotes**. These strings may include the following keywords:

BBE	Blackbox evaluations.
BBO	Blackbox outputs.
EVAL	Evaluations (includes cache hits).
MESH_INDEX	Mesh index ℓ [8].
MESH_SIZE	Mesh size parameter Δ_k^m [18].
OBJ	Objective function value.
POLL_SIZE	Poll size parameter Δ_k^p [18].
SGTE	Number of surrogate evaluations.
SIM_BBE	Simulated blackbox evaluations (includes initial cache hits).
SOL	Solution, with format <code>iSOLj</code> where <code>i</code> and <code>j</code> are two (optional) strings: <code>i</code> will be displayed before each coordinate, and <code>j</code> after each coordinate (except the last).
STAT_AVG	The <code>AVG</code> statistic (argument <code>STAT_AVG</code> of <code>BB_OUTPUT_TYPE</code>).
STAT_SUM	The <code>SUM</code> statistic defined by argument <code>STAT_SUM</code> for parameter <code>BB_OUTPUT_TYPE</code> .
TIME	Wall-clock time.
VARi	Value of variable <code>i</code> . The index 0 corresponds to the first variable.

In addition, all outputs may be formatted using the `C` style. Possibilities and examples are shown in the following table:

<code>%e</code>	Scientific notation (mantise/exponent) using <code>e</code> character.
<code>%E</code>	Scientific notation (mantise/exponent) using <code>E</code> character.
<code>%f</code>	Decimal floating point.
<code>%g</code>	Use the shorter of <code>%e</code> or <code>%f</code> .
<code>%G</code>	Use the shorter of <code>%E</code> or <code>%f</code> .
<code>%d</code> or <code>i</code>	Integer rounded value.

The number of columns (width) and the precision may also be indicated using also the `C` style as in the following examples:

format	width	precision
<code>%f</code>	auto	auto
<code>%5.4f</code>	5	4
<code>%5f</code>	5	auto
<code>%.4f</code>	auto	4
<code>%.f</code>	auto	0

For example, `'DISPLAY_STATS BBE & ($SOL,) & OBJ \\\'` displays lines similar to `'1 & (10.34 , 5.58) & -703.4734809 \\\'`, which may be copied into \LaTeX tables. A similar example with formatting may be `'DISPLAY_STATS BBE & ($%5.1fSOL,) & $%.2EOBJ$ \\\'` which gives `'1 & ($ 10.3$, $ 5.6$) & $-7.03E+02$ \\\'`. In case the user wants to explicitly display the `'%` character, it must be entered using `'\%'`.

Default values are `'DISPLAY_STATS BBE OBJ'` and `'DISPLAY_STATS OBJ'` for single and biobjective optimization, respectively (there is no need to enter `OBJ` twice in order for the two objective values to be displayed).

To write these outputs into the file `output.txt`, simply add the file name as first argument of `STAT_FILE`: for example `'STAT_FILE output.txt BBE (SOL) OBJ'`.

5.4.8 Fixed variables parameter `FIXED_VARIABLE`

This parameter is used to fix some variables to a value. This value is optional if at least one starting point is defined. The parameter may be entered with several types of arguments:

- A string indicating a text file containing n values. Variables will be fixed to the values that are not defined with the character ‘-’.
- A vector of n values with format (`v0 v1 ... vn-1`). Again, character ‘-’ may be used for free variables.
- An index range if at least one starting point has been defined (see 5.4.14 for practical examples of index ranges).
- An index range and a real value, with format ‘`FIXED_VARIABLE i-j v`’: variables i to j will be fixed to the value v ($i-j$ may be replaced by i).

5.4.9 Mesh and poll size parameters

The initial mesh size parameter Δ_0^m [18] is decided by `INITIAL_MESH_SIZE`. In order to achieve the scaling between variables, NOMAD considers the mesh size parameter as a vector of n elements. Note that a more explicit scaling method is available with the parameter `SCALING` (see Section 5.4.11). The same logic applies to the poll size parameter Δ_k^p . `INITIAL_MESH_SIZE` may be entered with the following formats:

- `INITIAL_MESH_SIZE d0`: initial mesh size for all variables.
- `INITIAL_MESH_SIZE (d0 d1 ... dn-1)`: for all variables (‘-’ may be used, and defaults will be considered).
- `INITIAL_MESH_SIZE i d0`: initial for variable i .
- `INITIAL_MESH_SIZE i-j d0`: initial for variables i to j .

The minimum mesh size Δ_{min}^m and the minimum poll size Δ_{min}^p (stopping criteria) may be defined the same way via parameters `MIN_MESH_SIZE` and `MIN_POLL_SIZE`. All values may also be preceded by ‘`r`’ to indicate a value relative to the bounds. For example, ‘`INITIAL_MESH_SIZE r0.1`’ means that $\Delta_0^m = (ub - lb)/10$ with $lb, ub \in \mathbb{R}^n$ and $lb \leq x \leq ub$ for all $x \in X$. Default is `r0.1` for variables with at least one bound. For an unbounded variable, the default value is taken as the maximum between 1 and the absolute value of the corresponding starting point coordinate.

5.4.10 Opportunistic strategy

The opportunistic strategy consists in terminating the evaluations of a list of trial points as soon as an improved value is found. This strategy is decided with the parameter `OPPORTUNISTIC_EVAL` and applies to both the poll and search steps. For the LH and Cache searches, the strategy may be chosen independently with `OPPORTUNISTIC_LH` and `OPPORTUNISTIC_CACHE_SEARCH`. If these parameters are not defined, the parameter `OPPORTUNISTIC_EVAL` applies to the LH and Cache searches. Other defaults are considered for biobjective optimization (see 6.2).

If the opportunistic strategy is enabled, some additional options may be defined via the following parameters:

- `OPPORTUNISTIC_MIN_NB_SUCCESS i`: do not terminate before i successes.

- `OPPORTUNISTIC_MIN_EVAL i`: do not terminate before `i` evaluations.
- `OPPORTUNISTIC_MIN_F_IMPRVMT r`: terminate only if f is reduced by $r\%$.
- `OPPORTUNISTIC_LUCKY_EVAL yes/no`: perform an additional blackbox evaluation after an improvement.

5.4.11 Scaling parameter `SCALING`

Scaling in NOMAD is automatically achieved via the mesh and poll size parameters which are vectors with one value per variable. However, this method relies on the existence of bounds. For the case when no bounds are available, or simply to give the user more control on the scaling, the parameter `SCALING` has been introduced in the version 3.4.

The parameter takes variable indices and values as arguments. During the algorithm, variables are multiplied by their associated value before an evaluation and the call to `NOMAD::Evaluator::eval_x()`. The variables are unscaled after the evaluation.

All NOMAD outputs (including files) display unscaled values. All variable-related parameters (bounds, starting points, fixed variables) must be specified without scaling. In a parameters file, the scaling is entered similarly to bounds or fixed variables. It is possible to specify a scaling for some variables and none for others. Enter the command `nomad.exe -h scaling` for more details about the use of `SCALING`.

5.4.12 Temporary directory parameter `TMP_DIR`

If NOMAD is installed on a network file system, with the batch mode use, the cost of read/write files will be high if no local temporary directory is defined. On `Linux/Unix/Mac OS X` systems, the directory `/tmp` is local and we advise the user to define '`TMP_DIR /tmp`'.

5.4.13 Group of variable parameter `VARIABLE_GROUP`

This parameter may be entered several times to define more than one group of variables. Variables in a group may be of different types (except for categorical variables). To define some particular types of directions or a particular Halton seed for this group, use the NOMAD library and `NOMAD::Parameters::set_VARIABLE_GROUP()`. In addition to the groups defined by parameters, NOMAD creates one group for all continuous, integer, and binary variables, and one group for categorical variables. If a group contains only binary variables, directions of type `NOMAD::GPS_BINARY` will be automatically used.

5.4.14 Starting point parameter `X0`

Parameter `X0` indicates the starting point of the algorithm. Several starting points may be proposed by entering this parameter several times. If no starting point is indicated, NOMAD considers the best evaluated point from an existing cache file (parameter `CACHE_FILE`) or from an initial Latin-Hypercube search (argument `p0` of `LH_SEARCH`). The `X0` parameter may take several types of arguments:

- A string indicating an existing cache file, containing several points (they can be already evaluated or not). This file may be the same as the one indicated with `CACHE_FILE`. If so, this file will be updated during the program execution, otherwise the file will not be modified.

- A string indicating a text file containing the coordinates of one or several points (values are separated by spaces or line breaks).
- n real values with format (v0 v1 ... vn-1).
- Two integers and one real:
 - ‘X0 i v’: (i+1)th coordinate set to v.
 - ‘X0 i-j v’: coordinates i to j set to v.
 - ‘X0 * v’: all coordinates set to v.
- One integer, another integer (or index range) and one real: the same as above except that the first integer k refers to the (k+1)th starting point.

The following example with $n = 3$ corresponds to the two starting points (5 0 0) and (-5 1 1):

X0 * 0.0
X0 0 5.0
X0 1 * 1.0
X0 1 0 -5.0

6 Special functionalities

6.1 Categorical variables

Categorical variables are discrete variables that can take a finite number of values. These are not integer or binary variables as there is no ordering property amongst the different values that can take the variables. A problem combining categorical variables with continuous variables or even ordinary discrete variables such as integer or binary is called a mixed variables optimization problem.

The algorithm used by NOMAD to handle mixed variables problems is defined in references [1, 4, 7, 15, 45] and works as follows.

6.1.1 Algorithm

At the end of an iteration where categorical variables are kept fixed, if no improvement has been made, a special step occurs, the *extended poll*. The extended poll first calls the user-provided procedure defining the neighborhood of categorical variables. The procedure returns a list of points that are neighbors of the current iterate such that categorical variables are changed and the other variables may or may not be changed. These points are called the *extended poll points* and their dimension may be different than the current iterate, for example when a categorical variable indicates the number of continuous variables.

The functions defining the problem are then evaluated at each of the extended poll points and the objective values are compared to the current best value. If the difference between the objective value at the current iterate and at an extended poll point is less than a parameter called the *extended poll trigger*, this extended poll point is called an *extended poll center* and a new MADS run is performed from this point. This run is called an *extend poll descent* and occurs on meshes that cannot be reduced more than the mesh of the beginning of the extended poll. If the opportunistic strategy is active, then the different extended poll descents are stopped as soon as a new success is achieved.

If surrogates are available, they can be used to evaluate the neighbors during the extended poll descent. The true functions will then be evaluated only on the most promising points. With surrogates, the extended poll costs at most the same number of true evaluations than the number of neighbors determined by the user-provided procedure.

6.1.2 Mixed variables optimization with NOMAD

We suggest the reader to follow this section along with the reading of the three examples located in `examples/advanced/categorical` that illustrate practical optimizations on mixed variables optimization problems.

In NOMAD, a categorical variable is identified by setting a `BB_INPUT_TYPE` parameter to the value 'C'. In addition, solving problems with categorical variables requires to define the neighbors of the current iterate. In batch mode, this is done by a separate executable (parameter `NEIGHBORS_EXE`) but with the limitation that the number of variables be the same than for the current iterate. See the provided example in `examples/advanced/categorical/batch` for such a case. The limitation of a fixed number of design variables is not present in library mode but requires user programming which is detailed in the remaining of this section.

Programming the method to define the categorical variables neighborhoods relies on a virtual method `NOMAD::Extended_Poll::construct_extended_points()` provided in NOMAD; the user must design its own `NOMAD::Extended_Poll` subclass in which `construct_extended_points()` is coded. This method takes as argument a point (the current iterate) and registers a list of extended poll points (the neighbors of the current iterate) by calling the method `NOMAD::Extended_Poll::add_extended_poll_point()`. In its main function, the user gives its own `NOMAD::Extended_Poll` object to the `NOMAD::Mads` object used to optimize the problem. If no `NOMAD::Extended_Poll` is provided to the `NOMAD::Mads` object, the program will generate an error.

In addition, each point in the algorithm possesses a signature (implemented in the `NOMAD::Signature` class), indicating the characteristics related to the variables: their number, their types, their bounds, their scaling, identification of fixed and periodic variables, and some information on the initial mesh size parameter for each variable. Hence, in the user-provided `NOMAD::Extended_Poll` subclass, for each extended poll point, a signature must be provided. If the extended poll point has the same characteristics than the current iterate, the signature of the current iterate can be used. However, if the number of variables varies according to the value taken by a categorical variable, a new signature must be created and the user is responsible for dealing with the associated memory allocations and deallocations. See the `NOMAD::Signature` class and the example located in `examples/advanced/categorical/single_obj/` for details about creating signatures.

Although the dimension of the problem may change during optimization, the starting points must all have the same characteristics (in particular number and types of variables). For these starting points, the `NOMAD::Parameters` class will automatically create a standard signature. However, if categorical variables are present the user must explicitly provide starting points. The reason is that the standard poll requires at least one starting point and an initial Latin-Hypercube search cannot be executed to find a starting point (see Section 5.4.14) because it has no reference signature for defining a value for each categorical variable.

The main parameter for mixed variable optimization is the extended poll trigger. Its value is indicated with the parameter `EXTENDED_POLL_TRIGGER`, and may be given as a relative value. The extended poll trigger is used to compare the objective values at an extended poll point y and at the current iterate x_k . If $f(y) < f(x_k) + \text{trigger}$, then y becomes an extended poll center from which a MADS run is performed. The default trigger value is `r0.2`, meaning that an extended poll point will become an extended poll center if $f(y)$ is less than $f(x_k) + f(x_k) \times 0.2$. See the

function `NOMAD::ExtendedPoll::check_trigger()` for the details of this test and for the cases where infeasible points or surrogate evaluations are considered.

Finally, please note that the boolean parameter `EXTENDED_POLL_ENABLED` can simply enable or disable the extended poll. When disabled, the categorical variables are simply fixed.

6.2 Biobjective optimization

NOMAD performs biobjective optimization through the BiMADS algorithm described in [26]. Handling of more than two objective functions will be implemented in future versions.

The BiMADS algorithm solves biobjective problems of the form

$$\min_{x \in \Omega} F(x) = (f_1(x), f_2(x)). \quad (2)$$

The algorithm launches successive runs of MADS on single-objective reformulations of the problem. An approximation of the Pareto front, or the list of points that are dominant following the definition of [26], is constructed with the evaluations performed during these MADS runs.

Two considerations must be taken into account when generating Pareto fronts: the quality of approximation of the dominant points and the repartition of these points. The quality of approximation may be measured with the `surf` criterion that gives the ratio of the area under the graph of the front relatively to a box enclosing all points (small values indicate a good front).

The quality of the coverage of the Pareto front is measured by the δ criterion, which corresponds to the largest distance between two successive Pareto points.

To define that a problem has two objectives, two arguments of the parameter `BB_OUTPUT_TYPE` must be set to `OBJ`. Then, NOMAD will automatically run the BiMADS algorithm. Additional parameters are:

- `MULTI_F_BOUNDS f1_min f1_max f2_min f2_max` (real values): these 4 values are necessary to compute the `surf` criterion. If not entered or if not valid (for example if `f1_min` is too big), then `surf` is not computed.
- `MULTI_FORMULATION` (string): single-objective reformulation [27]. This is how NOMAD computes one value from the two objective values. The argument must be in `{NORMALIZED, PRODUCT, DIST_L1, DIST_L2, DIST_LINF}` (`DIST_LINF` and `NORMALIZED` are equivalent). The default formulation is `PRODUCT` when `VNS` is not used, and `DIST_L2` otherwise.
- `MULTI_NB_MADS_RUNS` (integer): the number of single-objective MADS runs.
- `MULTI_OVERALL_BB_EVAL` (integer): the maximum number of blackbox evaluations over all MADS runs.
- `MULTI_USE_DELTA_CRIT` (bool, default to `no`): use or not a stopping criterion based on the δ measure.

Default values are considered if these parameters are not entered. All other MADS parameters are considered and apply to single MADS runs, with some adaptations:

- The `MAX_BB_EVAL` parameter corresponds to the maximum number of blackbox evaluations for one MADS run.
- The `F_TARGET` parameter is adapted to biobjective: it must be given with the two values z_1 and z_2 . If a point x is generated such that $f_1(x) \leq z_1$ and $f_2(x) \leq z_2$, then the algorithm terminates.

- Latin-Hypercube (LH) search (`LH_SEARCH p_0 p_1`): in single-objective optimization, `p_0` and `p_1` correspond to the initial number of search points and to the number of search points at each iteration, respectively. In the biobjective context, `p_0` is the number of initial search points generated in the first MADS run, and `p_1` is the number of points for the second MADS run. If no LH search is defined by the user, and if only `MULTI_OVERALL_BB_EVAL` is defined, then a default LH search is performed. Moreover, this default LH search is non-opportunistic (`OPPORTUNISTIC_LH` set to `no`).
- The parameter `SOLUTION_FILE` is disabled.

The NOMAD solution represents an approximation of the Pareto front and is accessible via the `DISPLAY_STATS` or `STATS_FILE` parameters. If `DISPLAY_DEGREE` is greater than 1, then the two measures `surf` and δ are displayed.

For a given budget of blackbox evaluations (`MULTI_OVERALL_BB_EVAL`), if the quality of approximation is desired (small value for `surf`), then single MADS optimizations must terminate after more severe criteria (for example a large number of blackbox evaluations, via `MAX_BB_EVAL`). If a better repartition of the points is desired (small value for δ), then the number of MADS runs should be larger, with less severe stopping criteria on single-objective optimizations.

6.3 Sensitivity analysis

Getting an optimizer is often insufficient for engineers. Two tools are available in the NOMAD package to perform sensitivity analyses for constraints, which is a useful tool to grasp more knowledge and see which constraints are important and which may be relaxed or tighten. What is generated by these tools is the data necessary to plot objective versus constraint graphs that help to understand the sensitivity. Details on the sensitivity analysis with blackboxes and some theoretical results on the smooth case may be consulted in [23].

The tools are available in directory `$NOMAD_HOME/tools/SENSITIVITY`. The first program is called `cache_inspect` and performs the *simple analysis* which consists in inspecting the cache produced after the execution of NOMAD on a constrained problem (the `CACHE_FILE` parameter must be set). The necessary inputs of this tool are a cache file and two blackbox output indices: one for the objective function, and one for the studied constraint. This last index may refer to a lower or an upper bound: in that case a file containing the bound values must be indicated. The program displays three columns with the values of the studied constraint $c_j(x)$ and of the objective $f(x)$, and a 0/1 flag indicating whether or not the couple $(c_j(x), f(x))$ is nondominated in the sense of the dominance notion of [26]. An optional parameter allows to display only nondominated points. These values may be plotted for example with a MATLAB script (one is available in the `cache_inspect` directory).

The second program, called `detailed_analysis`, performs the *detailed analysis*. With this tool, the original problem with constraint $c_j(x) \leq 0$ is replaced with the biobjective problem

$$\begin{aligned} \min_{x \in \Omega_j} \quad & (c_j(x), f(x)) \\ \text{s.t.} \quad & \underline{c}_j \leq c_j(x) \leq \bar{c}_j \end{aligned}$$

where Ω_j is the feasible set Ω minus the constraint. The use of the BiMADS algorithm allows to focus explicitly on the studied constraint in order to obtain a more precise sensitivity. The program takes as inputs a parameters file, the constraint and objective indices, and a cache file. The latter may be empty or not at the beginning of the execution, and it will be updated with the new evaluations. The updated cache file is in fact the output of the program and it may be

inspected with the `cache_inspect` tool in order to get the data for the sensitivity graphs. The \underline{c}_j and \bar{c}_j values used to bound the value of $c_j(x)$ may also be specified as input to the tool, as well as a maximum number of evaluations that bypasses the one inside the parameters file. Both programs may be executed without any input which result in the display of the required inputs description.

The typical way of using these tools is as follows: after a single run of MADS, the user uses the simple analysis in order to get a fast and free preview of the sensitivity. After that it is possible to get a more precise analysis on one or several constraints of interest using the detailed analysis, to the cost of additional evaluations.

6.4 Variable Neighborhood Search (VNS)

This search strategy is described in [12]. It is based on the Variable Neighborhood Search meta-heuristic [52, 53] as a search strategy to escape local minima. VNS should only be used for problems with several such local optima. It will cost some additional evaluations, since each search performs another MADS run from a perturbed starting point. Though, it will be a lot cheaper if a surrogate is provided via parameter `HAS_SGTE` or `SGTE_EXE`. We advise the user not to use VNS with biobjective optimization, as the BiMADS algorithm already performs multiple MADS runs.

In order to use the VNS search, which is disabled by default, the user has to define the parameter `VNS_SEARCH`, with a boolean or a real. This expected real value is the *VNS trigger*, which corresponds to the maximum desired ratio of VNS blackbox evaluations over the total number of blackbox evaluations. For example, a value of 0.75 means that NOMAD will try to perform a maximum of 75% blackbox evaluations within the VNS search. If a boolean is given as value to `VNS_SEARCH`, then a default of 0.75 is taken for the VNS trigger.

From a technical point of view, VNS is coded as a `NOMAD::Search` sub-class, and it is a good example of how a user-search may be implemented. See files `$NOMAD_HOME/src/VNS_Search.*pp` for details.

6.5 Parallel versions

Three parallel versions of the algorithm have been developed, namely P-MADS, COOP-MADS, and PSD-MADS. While P-MADS is directly implemented into NOMAD, the two others are programs using the NOMAD scalar library, and are located in the `tools` directory. These parallel versions are developed with MPI [55] under a master/slaves paradigm.

When creating blackbox problems it is important to keep in mind that the blackboxes will be called in parallel. So it is crucial that intermediary files possess different names: unique identifiers must be used. For that purpose, in library mode, in your custom `eval_x()` function, use the unique tag of the trial points with the method `NOMAD::Eval_Point::get_tag()`. It is also possible to use `NOMAD::get_pid()` to generate a unique identifier. In batch mode, NOMAD may communicate the seed and the tag of a point to the blackbox executable with the parameters `BB_INPUT_INCLUDE_SEED` and `BB_INPUT_INCLUDE_TAG` (see Section 5.2.2).

The user must be aware of the random aspect induced by the parallel versions. Even if deterministic directions such as ORTHOMADS are used, two parallel runs may not have the same outputs. Tests have suggested that P-MADS will give similar results than the scalar version, but much faster. The quality of the results may sometimes be less due to the fact that the usually efficient opportunistic strategy is not exploited as well as in the scalar version. However, the more evolved COOP-MADS strategy seems to give better results than the scalar version, and faster. The efficiency of the PSD-MADS algorithm is more noticeable on large problems (more than 20 and up to $\simeq 500$ variables) on which the other versions are not efficient.

A short description of the methods is given in the following sections, and for a more complete description as well as for numerical results, please consult [48].

6.5.1 The P-MADS method

P-MADS is the basic parallel version of the MADS algorithm where each list of trial points is simply evaluated in parallel. There are two versions of this method: first the **synchronous** version where an iteration is over only when all evaluations in progress are finished. With this strategy, some processes may be idle. The other version is the **asynchronous** method which consists in interrupting the iteration as soon a new success is made. If there are some evaluations in progress, these are not terminated. If these evaluations lead to successes after they terminate, then the algorithm will consider them and go back to these ‘old’ points. This version allows no process to be idle. The synchronous and asynchronous versions may be chosen via the parameter **ASYNCHRONOUS** whose default is **yes**.

The P-MADS executable is named `nomad.MPI.exe` and is located in the `bin` directory. It can be executed with the `mpirun` or `mpiexec` commands with the following format under **Linux**:

```
mpirun -np p $NOMAD_HOME/bin/nomad.MPI.exe param.txt
```

where `p` is the number of processes and `param.txt` is a parameters file with the same format as for the scalar version. If you have a number c of processors, then it is suggested to choose `np` to be equal to $c + 1$ (one master and c slaves). It may also be argued that `np` be proportional to the number of polling directions. For example, for a problem with $n = 3$ variables and $2n$ polling directions, each poll is going to generate 6 trial points, and on a 8-processors machine, choosing `np=7` may be a better choice than `np=9`.

6.5.2 The COOP-MADS method

The idea behind the COOP-MADS method is to run several MADS instances in parallel with different seeds so that no one has the same behavior.

A special process, called the cache server, replaces the usual master process. It implements a parallel version of the cache allowing each process to query if the evaluation at a given point has already been performed. This forbids any double evaluation. The cache server allows also the processes to perform the **cache search**, a special search consisting in retrieving, at each MADS iteration, the currently best known point.

The program given in the `tools` directory implements a simple version of the method where only one type of directions is used with different seeds: LT-MADS or ORTHOMADS, with a different random seed or a different Halton seed.

This program is not precompiled and the user must compile it as any other code using the NOMAD library. Makefiles for **X** systems and **Windows** are provided. Usage of the program is as follows:

```
mpirun -np p $NOMAD_HOME/tools/COOP-MADS/coopmads param.txt
```

as for P-MADS. Since the cache server is not demanding on computational time, the user can choose `np` to be the number of available processors plus one.

6.5.3 The PSD-MADS method

PSD-MADS corresponds to the parallel space decomposition of MADS described in [21]. The method aims at solving larger problems than the scalar version of NOMAD. While NOMAD is in general

efficient on problems for problems up to $\simeq 20$ variables, PSD-MADS has solved problems up to 500 variables.

In PSD-MADS, each slave process has the responsibility for a small number of variables on which a MADS algorithm is performed. These subproblems are decided by the master process. In the program given in the NOMAD package, as in the original paper, these groups of variables are chosen randomly, without any specific strategy. Concerning other aspects, the program given here is a simplified version of the one used for the SIOPT article. A cache server is also used as in COOP-MADS to forbid double evaluations. A special slave, called the pollster, works on all the variables, but with a reduced number of directions. The pollster ensures the convergence of the algorithm.

PSD-MADS must be compiled exactly as COOP-MADS, with the available makefile, and it executes with the command:

```
mpirun -np p $NOMAD_HOME/tools/PSD-MADS/psdmads param.txt bbe ns
```

where **bbe** is the maximal number of evaluations performed by each slave and **ns** is the number of variables considered by the slaves. So far, tests suggested that small values for these two parameters lead to good performance. In [21] and [48], **bbe**=10 and **ns**=2 are considered. The suggested strategy for **np** consists in setting it to the number of processors plus two (master and cache server are not demanding).

Future research include the design of evolved strategies in order to choose smart groups of variables on which slaves focus.

7 Release notes

7.1 Version 3.5

Version 3.5 adds the use of quadratic models to improve the software efficiency. Details and benchmarks are available in [33]. Mainly, two new features have been implemented:

- **Model search** (parameter `MODEL_SEARCH`): This is a new search strategy in which a local quadratic model is built and optimized in order to provide up to 4 new trial points at each iteration.
- **Model ordering** (parameter `MODEL_EVAL_SORT`): Before evaluating a list of trial points using the opportunistic strategy, a local quadratic model is built and the points are sorted accordingly to this model so that the most promising points are evaluated first.

Models are enabled by default, except with categorical variables and for problems with more than 50 variables. The model search is also disabled in parallel mode. The use of models usually improves the quality of the solution, but in the contrary they can be disabled by setting the value `no` to the 2 new parameters.

7.1.1 Minor changes

A series of bugs have been corrected in version 3.5.1 and some minor changes listed below have been applied.

- The new parameter `MAX_CONSECUTIVE_FAILED_ITERATIONS` allows to stop the algorithm after a number of unsuccessful iterations of the `Mads` algorithm.

- When no bounds are present, the initial mesh size (parameter `INITIAL_MESH_SIZE`) has a new default value: instead of being 1 it is now based on the coordinates of the starting point.
- The new parameter `NEIGHBORS_EXE` allows the handling of categorical variables in batch mode. See Section 6.1 and the example located in `examples/advanced/categorical/batch`.
- A series of parameters influencing the behavior of model search have been renamed for consistency and also to specify the type of model considered. `MODEL_MAX_TRIAL_PTS`, `MODEL_MAX_Y_SIZE`, `MODEL_MIN_Y_SIZE`, `MODEL_PROJ_TO_MESH`, `MODEL_RADIUS_FACTOR`, `MODEL_USE_WP` have been replaced respectively by `MODEL_SEARCH_MAX_TRIAL_PTS`, `MODEL_QUAD_MAX_Y_SIZE`, `MODEL_QUAD_MIN_Y_SIZE`, `MODEL_SEARCH_PROJ_TO_MESH`, `MODEL_QUAD_RADIUS_FACTOR`, `MODEL_QUAD_USE_WP`.
- When CTRL-C is pressed an evaluation can be interrupted in library mode within the user provided function `eval_x()` (see Section 4). This is achieved by the following test: `if (NOMAD::Evaluator::get_force_quit()) {...}`.
- A random number generator have been implemented to allow repeatability of the results on different platforms.
- A bug in the display format of the stats present when compiling with Visual Studio C++ has been corrected (hexadecimal display).
- A bug when using categorical variables with varying problem dimensionality has been fixed.
- A bug in the values of integers for fine meshes has been fixed.
- A bug in the display stats for the phase one search has been corrected.

7.2 Previous versions

7.2.1 Version 3.4

- **Parallelism:** Three parallel algorithms are now available. See Section 6.5 for details.
- All NOMAD types and classes are now included in the namespace `NOMAD`. Consequently enumeration types and constants have their names changed from `_X_` to `NOMAD::X`.
- A documentation has been constructed in the HTML format with the [doxygen](#) documentation generator. It is available from the [NOMAD website](#) at www.gerad.ca/nomad/doxygen/html.
- NOMAD is now distributed under the GNU Lesser General Public License (LGPL). The license can be found as a text file in the `src` directory or at www.gnu.org/licenses.
- A new parameter `SCALING` allowing the scaling of the variables. See Section 5.4.11.
- Tool for sensitivity analysis (see Section 6.3).

7.2.2 Version 3.3

- Handling of **categorical variables** for mixed variable problems (MVP). See Section 6.1.

7.2.3 Version 3.2

- **Variable Neighborhood Search** (VNS) described in Section 6.4.
- **Installers** for X systems.
- **Help on parameters** included in the executable: the command ‘`nomad -h keyword`’ displays help on the parameters related to `keyword`. Typing only ‘`nomad -h`’ or ‘`nomad -help`’ displays all the available help: a complete description of all parameters. Also, ‘`nomad -i`’ or ‘`nomad -info`’ displays information on the current release, and ‘`nomad -v`’ displays the current version.

7.2.4 Version 3.1

- **Biobjective optimization**: see Section 6.2.
- **Periodic variables**: if some variable are periodic, this may be indicated via parameter `PERIODIC_VARIABLE`. Bounds must be defined for these variables. The MADS algorithm adapted to periodic variables is described in [24].
- **Groups of variables** can be defined with the parameter `VARIABLE_GROUP`. At every MADS poll, different directions will be generated for each group. For example, for a location problem, if groups correspond to spatial objects, these will be moved one at a time.

7.3 Future versions

Future algorithmic developments include:

- Adaptive surrogates and use of the surrogate management framework [31].
- MULTI-MADS: multi-objective variant of MADS [26], with 3 and more objective functions.
- Use of simplex gradients [35, 36].

Acknowledgments

The developers of NOMAD wish to thank Florian Chambon, Mohamed Sylla and Quentin Reynaud, all from [ISIMA](#), for their contribution to the project during Summer internships, and to Anthony Guillou and Dominique Orban for their help with `configure` and AMPL, and their suggestions. Thanks also to Annie Angers, Maud Bay, Eve Bélisle, Vincent Garnier, Michal Kvasnicka, Alexander Lutz, Rosa-Maria Torres-Calderon, Yuri Vilmanis, Martin Posch and an anonymous user for their tests, feedback, and various help. Finally, many thanks to the TOMS anonymous referees for their useful comments which helped a lot to improve the code and the text of [47].

Related publications

- [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, J.W. Chrissis, and J.G. Walston. Mesh adaptive direct search algorithms for mixed variable optimization. *Optimization Letters*, 3(1):35–47, 2009.
- [5] M.A. Abramson, C. Audet, G. Couture, J.E. Dennis, Jr., and S. Le Digabel. The NOMAD project. Software available at <http://www.gerad.ca/nomad>.
- [6] M.A. Abramson, C. Audet, and J.E. Dennis, Jr. Generalized pattern searches with derivative information. *Mathematical Programming*, Series B, 100:3–25, 2004.
- [7] M.A. Abramson, C. Audet, and J.E. Dennis, Jr. Filter pattern search algorithms for mixed variable constrained optimization problems. *Pacific Journal of Optimization*, 3(3):477–500, 2007.
- [8] M.A. Abramson, C. Audet, J.E. Dennis, Jr., and S. Le Digabel. OrthoMADS: A deterministic MADS instance with orthogonal directions. *SIAM Journal on Optimization*, 20(2):948–966, 2009.
- [9] M.A. Abramson, O.A. Brezhneva, J.E. Dennis Jr., and R.L. Pingel. Pattern search in the presence of degenerate linear constraints. *Optimization Methods and Software*, 23(3):297–319, 2008.
- [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, 2008.
- [12] C. Audet, V. Béchar, and S. Le Digabel. Nonsmooth optimization through mesh adaptive direct search and variable neighborhood search. *Journal of Global Optimization*, 41(2):299–318, 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, 2001.
- [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. A pattern search filter method for nonlinear programming without derivatives. *SIAM Journal on Optimization*, 14(4):980–1010, 2004.
- [18] C. Audet and J.E. Dennis, Jr. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization*, 17(1):188–217, 2006.

- [19] C. Audet and J.E. Dennis, Jr. Nonlinear programming by mesh adaptive direct searches. *SIAG/Optimization Views-and-News*, 17(1):2–11, 2006.
- [20] C. Audet and J.E. Dennis, Jr. A progressive barrier for derivative-free nonlinear programming. *SIAM Journal on Optimization*, 20(4):445–472, 2009.
- [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, J.E. Dennis, Jr., and S. Le Digabel. Globalization strategies for mesh adaptive direct search. *Computational Optimization and Applications*, 46(2):193–215, 2010.
- [23] C. Audet, J.E. Dennis, Jr., and S. Le Digabel. Trade-off studies in blackbox optimization. Technical Report G-2010-49, Les cahiers du GERAD, 2010. To appear in *Optimization Methods and Software*.
- [24] 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. To appear in *Pacific Journal of Optimization*.
- [25] C. Audet and D. Orban. Finding optimal algorithmic parameters using derivative-free optimization. *SIAM Journal on Optimization*, 17(3):642–664, 2006.
- [26] 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.
- [27] C. Audet, G. Savard, and W. Zghal. A mesh adaptive direct search algorithm for multiobjective optimization. *European Journal of Operational Research*, 204(3):545–556, 2010.
- [28] 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, 2007.
- [29] 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.
- [30] 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.
- [31] 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 and Multidisciplinary Optimization*, 17(1):1–13, 1999.
- [32] A. Brooke, D. Kendrick, and A. Meeraus. *GAMS: A Users’ Guide*. The Scientific Press, Danvers, Massachusetts, 1988.
- [33] A.R. Conn and S. Le Digabel. Use of quadratic models with mesh adaptive direct search for constrained black box optimization. Technical Report G-2011-11, Les cahiers du GERAD, 2011. To appear in *Optimization Methods and Software*.

- [34] 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.
- [35] 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.
- [36] A.L. Custódio and L.N. Vicente. Using sampling and simplex derivatives in pattern search methods. *SIAM Journal on Optimization*, 18(2):537–555, 2007.
- [37] 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, 2004.
- [38] J.E. Dennis, Jr. and V. Torczon. Direct search methods on parallel machines. *SIAM Journal on Optimization*, 1(4):448–474, 1991.
- [39] 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, 2008.
- [40] A.E. Gheribi, C. Audet, S. Le Digabel, E. Bélisle, C.W. Bale, and A.D. Pelton. Calculating optimal conditions for alloy and process design using thermodynamic and properties databases, the factsage software and the mesh adaptive direct search (MADS) algorithm. Technical Report G-2010-77, Les cahiers du GERAD, 2011. To appear in *CALPHAD: Computer Coupling of Phase Diagrams and Thermochemistry*.
- [41] A.E. Gheribi, C. Robelin, S. Le Digabel, C. Audet, and A.D. Pelton. Calculating all local minima on liquidus surfaces using the factsage software and databases and the mesh adaptive direct search algorithm. *The Journal of Chemical Thermodynamics*, 43(9):1323–1330, 2011.
- [42] 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.
- [43] 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.
- [44] L.A. Sweatlock K. Diest and D.E. Marthaler. Metamaterials design using gradient-free numerical optimization. *Journal of Applied Physics*, 108(8):1–5, 2010.
- [45] 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.
- [46] S. Le Digabel. NOMAD user guide. Technical Report G-2009-37, Les cahiers du GERAD, 2009.
- [47] S. Le Digabel. Algorithm 909: NOMAD: Nonlinear optimization with the MADS algorithm. *ACM Transactions on Mathematical Software*, 37(4):44:1–44:15, 2011.

- [48] S. Le Digabel, M.A. Abramson, C. Audet, and J.E. Dennis, Jr. Parallel versions of the MADS algorithm for black-box optimization. In *Optimization days*, Montreal, May 2010. GERAD. Slides available at www.gerad.ca/Sebastien.Le.Digabel/talks/2010_JOPT_25mins.pdf.
- [49] 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.
- [50] 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.
- [51] 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, 2007.
- [52] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers and Operations Research*, 24(11):1097–1100, 1997.
- [53] P. Hansen N. Mladenović. Variable neighborhood search: principles and applications. *European Journal of Operational Research*, 130(3):449–467, 2001.
- [54] M.S. Ouali, H. Aoudjit, and C. Audet. Optimisation des stratégies de maintenance. *Journal Européen des Systèmes Automatisés*, 37(5):587–605, 2003.
- [55] M. Snir, S.W. Otto, S. Huss-Lederman, D.W. Walker, and J. Dongarra. *MPI: The Complete Reference*. The MIT Press, Cambridge, Massachusetts, 1995.
- [56] T.A. Sriver, J.W. Chrissis, and M.A. Abramson. Pattern search ranking and selection algorithms for mixed variable stochastic optimization, 2004. Preprint.
- [57] R. Torres, C. Bès, J. Chaptal, and J.-B. Hiriart-Urruty. Optimal, environmentally-friendly departure procedures for civil aircraft. *Journal of Aircraft*, 48(1):11–22, 2011.