Nomad 4

C. Tribes and V. Rochon Montplaisir

Jul 07, 2021

INTRODUCTION:

Warning: This user guide is specific to NOMAD 4.

NOMAD 3 is still available. It will be replaced by NOMAD 4 in the future.

Get NOMAD 3 and 4 at https://www.gerad.ca/nomad/.

NOMAD is a blackbox optimization software. A general presentation of NOMAD is given in *Introduction*.

New users of NOMAD should refer to

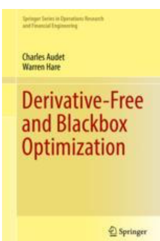
- *Installation*
- *Getting started*

Using NOMAD

- Starting from *NOMAD usage*, all users can find ways to tailor problem definition, algorithmic settings and software output.
- Refer to *Advanced functionalities* and *Tricks of the trade* for specific problem solving.

Please cite NOMAD 4 with reference:

Reference book



A complete introduction to derivative-free and blackbox optimization can be found in the textbook:

CHAPTER ONE

INTRODUCTION

Note: NOMAD = Nonlinear Optimization by Mesh Adaptive Direct Search

NOMAD is a software application for simulation-based optimization. It can efficiently explore a design space in search of better solutions for a large spectrum of optimization problems. NOMAD is at its best when applied to blackbox functions.

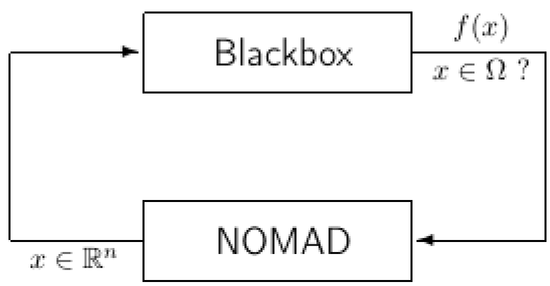


Fig. 1: Blackbox optimization

Such functions are typically the result of expensive computer simulations which

- have no exploitable property such as derivatives,
- may be contaminated by noise,
- may fail to give a result even for feasible points.

NOMAD is a C++ implementation of the **Mesh Adaptive Direct Search (MADS)** algorithm (see references [AbAuDeLe09], [AuDe2006], [AuDe09a] for details) designed for constrained optimization of blackbox functions in the form

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

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

1.1 Basics of the MADS algorithm

At the core of NOMAD resides the *Mesh Adaptive Direct Search (MADS)* algorithm. As the name implies, this method generates iterates on a series of meshes with varying size. A mesh is a discretization of the space of variables. However, also as the name implies, the algorithm performs an adaptive search on the meshes including controlling the refinement of the meshes. The reader interested in the rather technical details should read Reference [AuDe2006].

The objective of each iteration of the *MADS* algorithm, is to generate a trial point on the mesh that improves the current best solution. When an iteration fails to achieve this, the next iteration is initiated on a finer mesh.

Each iteration is composed of two principal steps called the *Search* and the *Poll* steps [AuDe2006]. The *Search* step is crucial in practice because it is so flexible and can improve the performance significantly. The *Search* step is constrained by the theory to return points on the underlying mesh, but of course, it is trying to identify a point that improves the current best solution.

The *Poll* step is more rigidly defined, though there is still some flexibility in how this is implemented. The *Poll* step generates trial mesh points in the vicinity of the best current solution. Since the *Poll* step is the basis of the convergence analysis, it is the part of the algorithm where most research has been concentrated.

A high-level presentation of *MADS* is shown in the pseudo-code below.

Algorithm 1: High-level presentation of MADS

Initialization: Let $x_0 \in \mathbb{R}^n$ be an initial point and set the iteration counter $k \leftarrow 0$
Main loop:
repeat
  SEARCH on the mesh to find a better solution than x_k
  if *the* SEARCH *failed* **then**
    POLL on the mesh to find a better solution than x_k
  if *a better solution than x_k was found by either the* SEARCH *or the* POLL **then**
    call it x_{k+1} and coarsen the mesh
  else
    set $x_{k+1} = x_k$ and refine the mesh
  Update parameters and set $k \leftarrow k + 1$
until *Stopping criteria is satisfied*;

1.2 Using NOMAD

Warning: NOMAD does not provide a graphical user interface to define and perform optimization.

Minimally, users must accomplish several tasks to solve their own optimization problems:

- Create a custom blackbox program(s) to evaluate the functions f and c_j OR embed the functions evaluations in C++ source code to be linked with the NOMAD library.
- Create the optimization problem definition in a parameter file OR embed the problem definition in C++ source code to be linked with the NOMAD library.
- Launch the execution at the command prompt OR from another executable system call.

Users can find several examples provided in the installation package and described in this user guide to perform customization for their problems. The installation procedure is given in *Installation*. New users should refer to *Getting started*. The most important instructions to use NOMAD are in :ref:'basic_nomad_usage'. In addition, tricks that may help solving specific problems and improve NOMAD efficiency are presented in *Tricks of the trade*. Advanced parameters and functionalities are presented in *Advanced functionalities*.

1.3 Supported platforms and environments

NOMAD source codes are in C++ and are identical for all supported platforms. See *Installation* for details to obtain binaries from the source files.

1.4 Authors and fundings

The development of NOMAD started in 2001. Three versions of NOMAD have been developed before NOMAD 4. NOMAD 4 and NOMAD 3 are currently supported. NOMAD 4 is almost a completely new code compared with NOMAD 3.

NOMAD 4 has been funded by Huawei Canada, Rio Tinto, Hydro-Québec, NSERC (Natural Sciences and Engineering Research Council of Canada), InnovÉÉ (Innovation en Énergie Électrique) and IVADO (The Institute for Data Valorization)

NOMAD 3 was created and developed by Charles Audet, Sebastien Le Digabel, Christophe Tribes and Viviane Rochon Montplaisir and was funded by AFOSR and Exxon Mobil.

NOMAD 1 and 2 were created and developed by Mark Abramson, Charles Audet, Gilles Couture, and John E. Dennis Jr., and were funded by AFOSR and Exxon Mobil.

The library for dynamic surrogates (SGTELIB) has been developed by Bastien Talgorn (bastien-talgorn@fastmail.com), McGill University, Montreal. The SGTELIB is included in NOMAD since version 3.8.0.

Developers of the methods behind NOMAD include:

- Mark A. Abramson (abramson@mathematics.byu.edu), Bringham Young University.
- Charles Audet (https://www.gerad.ca/Charles.Audet), GERAD and Département de mathématiques et de génie industriel, École Polytechnique de Montréal.
- J.E. Dennis Jr. (http://www.caam.rice.edu/\protect\begingroup\immediate\write\@unused\def\MessageBreak˙\let\protect\edefYourcommandwasignored.\MessageBreakTypeI<command><return>toreplaceitwithanothercommand,\MessageBreakor<return>tocontinuewithoutit.\errhelp\let\def\MessageBreak˙(inputenc)\def\errmessagePackageinputencError:UnicodecharâĹij(U+223C)\MessageBreaknotsetupforusewithLaTeX.˙˙Seetheinputencpackagedocumentationforexplanation.˙TypeH<return>forimmediatehelp\endgroupdennis), Computational and Applied Mathematics Department, Rice University.
- Sébastien Le Digabel (http://www.gerad.ca/Sebastien.Le.Digabel), GERAD and Département de mathématiques et de génie industriel, École Polytechnique de Montréal.
- Viviane Rochon Montplaisir, GERAD (https://www.gerad.ca/en/people/viviane-rochon-montplaisir) and Département de mathématiques et de génie industriel, École Polytechnique de Montréal.
- Christophe Tribes, GERAD (https://www.gerad.ca/en/people/christophe-tribes) and Département de mathématiques et de génie industriel, École Polytechnique de Montréal.

1.5 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 AMPL, and their suggestions.

A special thank to Maud Bay, Eve Bélisle, Vincent Garnier, Michal Kvasnička, Alexander Lutz, Rosa-Maria Torres-Calderon, Yuri Vilmanis, Martin Posch, Etienne Duclos, Emmanuel Bigeon, Walid Zghal, Jerawan Armstrong, Stéphane Alarie and Klaus Truemper for their feedbacks and tests that significantly contributed to improve NOMAD. Some features of NOMAD have been developed under the impulsion of enthusiastic users/developers: Andrea Ianni, Florian Chambon, Mohamed Sylla, Quentin Reynaud, Amina Ihaddadene, Bastien Talgorn, Nadir Amaioua and Catherine Poissant. We also wish to thank Pascal Côté for his contribution in the development of the Python interface pyNomad and Jonathan Currie for the development of the foundations for a strong NOMAD interface for MATLAB.

The contributions of Miguel Anjos, Romain Couderc, Miguel Diago Martinez, Solène Kojtych, Guillaume Lameynardie, Wim Lavrijsen, Alexis Montoison, Caroline Rocha, Ludovic Salomon and Renaud Saltet was highly appreciated during the development and testing of NOMAD 4.

CHAPTER

TWO

LICENSE

NOMAD is a free software application released under the GNU Lesser General Public License v 3.0. As a free software application you can redistribute and/or modify NOMAD source codes under the terms of the GNU Lesser General Public License.

For more information, please refer to the local copy of the license obtained during installation. For additional information you can contact us or visit the Free Software Foundation website.

CHAPTER

THREE

CONTACT US

All queries can be submitted by email at

Note: nomad@gerad.ca.

In particular, feel free to ask technical support for problem specification (creating parameter files or integration with various types of simulations) and system support (installation and plateform-dependent problems).

Bug reports and suggestions are valuable to us! We are committed to answer to posted requests as quickly as possible.

References

CHAPTER

FOUR

INSTALLATION

On Linux, Windows and Mac OS X, NOMAD can be compiled using *CMake*, a tool to manage building of source code.

The minimum version of *CMake* is 3.14. Older versions should trigger an error.

A recent C++ compiler supporting C++14 is also required. The compilation has been tested on Linux with gcc 9.3.0, 10.1.0 and 11.1.0. The compilation has been tested on OSX with gcc Homebrew 9.3.0 and Apple clang version 11.0.3. The compilation has been tested on Windows 8 with Microsoft Visual Studio 2019 (cl.exe 19.29.300038.1) and Microsoft Visual Studio 2017.

CMake will detect which compiler is available.

Warning:

Some older version of *CMake* do not trigger an explicit error on the version number. If the `cmake` commands fail, check the version manually on the command line

```
cmake --version
```

The minimum acceptable version is 3.14.

Note: If the version of *CMake* is older than 3.14 or if you do not have *CMake* installed, we recommend to install *CMake* using a **package manager**. The other option is to follow the procedure given at cmake.org to obtain binaries.

For Mac OSX, *CMake* can be installed on the command line using package manager MacPorts or Homebrew.

For Linux, several package managers exist to automate the procedure.

For Windows, an installer tool is available at cmake.org/download. Please note that all commands are performed in the Windows Command Prompt windows of Visual Studio.

The NOMAD installation procedure has the three following steps: **configuration, building and installation**.

Warning: Before starting the procedure we recommend to set the environment variable `$NOMAD_HOME` with the path where NOMAD has been copied. For Linux and OSX,

```
export NOMAD_HOME=/home/myUserName/PathToNomad
```

For Windows, add an environment variable `%NOMAD_HOME%` containing the path.

The remaining of the documentation uses the `$NOMAD_HOME` environment variable.

4.1 1- Configuration using provided `CMakeLists.txt` files

On the command line, in the `$NOMAD_HOME` directory:

```
cmake -S . -B build/release
```

Building options

To enable time stats build:

```
cmake -DTIME_STATS=ON -S . -B build/release
```

To enable interfaces (C and Python) building:

```
cmake -DBUILD_INTERFACES=ON -S . -B build/release
```

To disable *OpenMP* compilation:

```
cmake -DTEST_OPENMP=OFF -S . -B build/release
```

This command creates the files and directories for building (`-B`) in `build/release`. The source (`-S`) `CMakeLists.txt` file is in the `$NOMAD_HOME` directory.

The command can be modified to enable/disable some options (see side bar).

OpenMP is used for parallelization of evaluations. *CMake* will detect if *OpenMP* is available by default. To forcefully deactivate compilation with *OpenMP*, see option in side bar.

4.2 2- Build

Build the libraries and applications (Linux/OSX):

```
cmake --build build/release
```

For Windows, the default configuration is Debug. To obtain the Release version:

```
cmake --build build/release --config Release
```

Option `--parallel xx` can be added for faster build

It is possible to build only a single application in its working directory:

```
cd $NOMAD_HOME/examples/basic/library/example1
cmake --build $NOMAD_HOME/build/release --target example1_lib.exe
```

4.3 3- Install

Copy binaries and headers in build/release/[bin, include, lib] and in the examples/tests directories:

```
cmake --install build/release
```

Option –config Release should be used on Windows to install Release configuration.

The executable `nomad` will installed into the directory:

```
$NOMAD_HOME/build/release/bin/
```

Additionally a symbolic link to `nomad` binary is available:

```
$NOMAD_HOME/bin
```

4.4 Bulding for debug version

The procedure to configure, build and install the `debug` version is the following (linux/OSX). On the command line in the `$NOMAD_HOME` directory:

```
cmake -S . -B build/debug -D CMAKE_BUILD_TYPE=Debug

cmake --build build/debug

cmake --install build/debug
```

On Windows, all 4 configurations are always build Debug, RelWithDebugInfo, MinSizeRel, Release); the flag CMAKE_BUILD_TYPE can be ignored.

4.5 Use another compiler

The environment variables `CC` and `CXX` can be used to select the `C` and `C++` compilers.

Note: `Clang` is the default compiler for Mac OSX using XCode. But, *OpenMP* (used for parallel evaluations) support is disabled in *Clang* that come with *Xcode*. Users of Mac OSX can install and use another compiler to enable *OpenMP* support. For example, `GCC` compilers can be obtained using MacPorts or Homebrew.

CHAPTER FIVE

TESTING INSTALLATION

Once building **and installation** have been performed some tests can be performed. By default the examples are built and can be tested:

The NOMAD binary can be tested:

```
$NOMAD_HOME/bin/nomad -v
```

This should return the version number on the command line.

Additionally, by default the examples are built and can be tested:

```
cd build/release
ctest
```

Option `--parallel xx` can be added for faster execution. The log of the tests can be found in `$NOMAD_HOME/build/release/Testing/Temporary`.

CHAPTER

SIX

GETTING STARTED

NOMAD is an efficient tool for simulation-based design optimizations provided in the form:

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

where the feasible set $\Omega = \{x \in X : c_j(x) \leq 0, j \in J\} \subset \mathbb{R}^n$, $f, c_j : X \rightarrow \mathbb{R} \cup \{\infty\}$ for all $j \in J = \{1, 2, \ldots, m\}$, and where X is a subset of $\mathbb{R}^n$. The functions f and c_j, jJ, are typically blackbox functions whose evaluations require computer simulation.

NOMAD can be used in two different modes: batch mode and library mode. The batch mode is intended for a basic usage and is briefly presented below (more details will be provided in *NOMAD usage*). This chapter explains how to get started with NOMAD in batch mode. The following topics will be covered:

- *Create blackbox programs*
- *Provide parameters* for defining the problem and displaying optimization results
- *Conduct optimization*

Note: Building NOMAD binaries and running the examples provided during the installation requires to have a `C++` compiler installed on your machine.

Compilation instructions rely on **CMake** and have been tested with **GCC** (GNU Compiler Collection) on Linux and OSX.

6.1 Create blackbox programs

To conduct optimization in batch mode the users must define their separate blackbox program coded as a standalone program. Blackbox program executions are managed by NOMAD with system calls.

A valid blackbox program:

- takes an input vector file as single argument,
- reads space-separated values in input vector file,
- returns evaluation values on standard output or file,
- returns an evaluation status.

In what follows we use the example in the `$NOMAD_HOME/examples/basic/batch/single_obj`. This example optimization problem has a single objective, 5 variables, 2 nonlinear constraints and 8 bound constraints:

$$\min_{x \in \mathbb{R}^5} f(x) = x_5$$

$$\text{subject to} \quad \begin{cases} c_1(x) = \sum_{i=1}^{5} (x_i - 1)^2 - 25 & \leq 0 \\ c_2(x) = 25 - \sum_{i=1}^{5} (x_i + 1)^2 & \leq 0 \\ \qquad\qquad\qquad\qquad x_i & \geq -6 \quad i = 1, 2, \ldots, 5 \\ \qquad\qquad\qquad\qquad x_1 & \leq 5 \\ \qquad\qquad\qquad\qquad x_2 & \leq 6 \\ \qquad\qquad\qquad\qquad x_3 & \leq 7\,. \end{cases}$$

Note: The blackbox programs may be coded in any language (even scripts) but must respect **NOMAD format**:

1. The blackbox program must be callable in a terminal window at the command prompt and take the input vector file name as a single argument. For the example above, the blackbox executable is `bb.exe`, one can execute it with the command `./bb.exe x.txt`. Here `x.txt` is a text file containing a total of 5 values.

2. NOMAD will manage the creation of the **input file consisting of one value for each variable separated by space** and the execution of the blackbox program.

3. The blackbox program must return the evaluation values by displaying them in the **standard output** (default) or by writing them in an output file (see *Advanced functionalities* about advanced output options). It must also **return an evaluation status of 0** to indicate that the evaluation went well. Otherwise NOMAD considers that the evaluation has failed.

4. The minimum 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(s) that one seeks to minimize. The constraints values correspond to left-hand side of 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 blackbox C++ program of the previous example to evaluate the objective and the two constraints for a given design vector is given as:

```
#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++ ) {
```

(continues on next page)

(continued from previous page)

```
        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;
}
```

The blackbox compilation and test are as follows:

1. Change directory to `$NOMAD_HOME/examples/basic/batch/single_obj`.
2. Optionally, compile the blackbox program with the following command `g++ -o bb.exe bb.cpp` (**GNU compiler**). This step is not really required because the building procedure with *CMake* normally builds the blackbox executable for this example.
3. Test the executable with the text file `x.txt` containing `0 0 0 0 0` by entering the command `bb.exe x.txt`.

4. 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 not satisfied ($c_2(x) = 20 > 0$).

```
> cd $NOMAD_HOME/examples/basic/batch/single_obj
> g++ -o bb.exe bb.cpp
> more x.txt
0 0 0 0 0
> ./bb.exe x.txt
0 -20 20
```

Note: The order of the displayed outputs must correspond to the order defined in the parameter file (see *BB_OUTPUT_TYPE* for details). If variables have bound constraints, they must be defined in the parameters file and should not appear in the blackbox code.

6.2 Provide parameters

In batch mode, the parameters are provided in a text file using predefined keywords followed by one or more argument.

Note: Help on parameters is accessible at the command prompt: `$NOMAD_HOME/bin/nomad -h param_name`

Here are some of the most important parameters defining an optimization problem (without brackets):

- The number of variables (`DIMENSION n`).

- The name of the blackbox executable that outputs the objective and the constraints (`BB_EXE bb_name`).
- Bounds on variables are defined with the `LOWER_BOUND lb` and `UPPER_BOUND ub` parameters.
- The output types of the blackbox executable: objective and constraints (`BB_OUTPUT_TYPE obj cons1 ... consM`).
- A starting point (`X0 x0`).
- An optional stopping criterion (`MAX_BB_EVAL max_bb_eval`, for example). If no stopping criterion is specified, the algorithm will stop as soon as the mesh size reaches a given tolerance.
- Any entry on a line is ignored after the character `#`.

Note: The order in which the parameters appear in the file or their case is unimportant.

Example of a basic parameters file extracted from `$NOMAD_HOME/examples/basic/batch/single_obj/param.txt`. The comments in the file describes some of the syntactic rules to provide parameters:

```
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 black-box evaluations have
                              # been made
```

The constraints defined in the parameters file are of different types. The first constraint $c_1(x) \le 0$ is treated by the *Progressive Barrier* approach (*PB*), which allows constraint violations. The second constraint, $c_3(x) \le 0$, is treated by the *Extreme Barrier* approach (*EB*) that forbids violations. Hence, evaluations not satisfying extreme barrier constraints are simply not considered when trying to improve the solution.

In the example above, the algorithmic parameters of NOMAD need not to be set because default values are considered. This will provide the best results in most situations.

6.3 Conduct optimization

Optimization is conducted by starting NOMAD executable in a command window with the parameter file name given as argument.

```
$NOMAD_HOME/bin/nomad param.txt
```

To illustrate the execution, the example provided in `$NOMAD_HOME/examples/basic/batch/single_obj/` is considered:

```
> cd $NOMAD_HOME/examples/basic/batch/single_obj
> ls
bb.cpp bb.exe CMakeLists.txt makefile param.txt x.txt
>$NOMAD_HOME/bin/nomad param.txt
BBE ( SOL ) OBJ
  1   (   0          0          0          0          0         )   0         (Phase One)
  8   (   0          4          0          0          0         )   0         (Phase One)
  28  (   1.4        5          0         -0.6       -0.4       )  -0.4
  29  (   2.6        4          0         -1.4       -0.8       )  -0.8
  33  (   1.63       3          0.92      -1.78      -0.88      )  -0.88
  37  (   2.46       3          0.97      -1.87      -0.92      )  -0.92
  41  (   3.2        3          0.16      -1.26      -1.05      )  -1.05
  42  (   4.27       2         -0.23      -1.07      -1.36      )  -1.36
  47  (   3.0        1          1.22      -1.92      -1.5       )  -1.5
  48  (   3.2        0          1.83      -2.19      -1.86      )  -1.86
  57  (   3.91      -0          1.02      -1.32      -1.95      )  -1.95
  67  (   3.61      -0          1.28      -1.83      -1.99      )  -1.99
  78  (   3.94       1          0.63      -1.14      -2.02      )  -2.02
  79  (   4.32       1          0.02      -0.61      -2.11      )  -2.11
  84  (   3.68       0          0.97      -1.23      -2.15      )  -2.15
  88  (   3.91       1          0.5       -0.6       -2.2       )  -2.2
  89  (   4.07       1          0.1        0.01      -2.31      )  -2.31
  94  (   3.67       1          0.56      -0.47      -2.36      )  -2.36
  95  (   3.35       1          0.84      -0.39      -2.48      )  -2.48
  99  (   4.15       1         -0.37       0.57      -2.49      )  -2.49
  Reached stop criterion: Max number of blackbox evaluations (Eval Global) 100
  A termination criterion is reached: Max number of blackbox evaluations (Eval Global)␣
↪No more points to evaluate 100

  Best feasible solution:     #1540 ( 4.15 1 -0.37 0.57 -2.49 )   Evaluation OK    f =  -
↪2.4900000000000002132     h =   0

  Best infeasible solution:   #1512 ( 3.79 0 1.14 -1.75 -1.97 )   Evaluation OK    f =  -
↪1.9699999999999999734     h =   0.0350064099999999475

  Blackbox evaluations:        100
  Total model evaluations:     1348
  Cache hits:                  3
  Total number of evaluations: 103
```

CHAPTER

SEVEN

NOMAD USAGE

This chapter describes how to use NOMAD for solving blackbox optimization problems. Functionalities of NOMAD that are considered more advanced such as parallel evaluations are presented in *Advanced functionalities*.

Note: New users are encouraged to first read *Getting started* to understand the basics of NOMAD utilization.

Note: Many examples are provided in `$NOMAD_HOME/examples` with typical optimization outputs.

Batch mode is presented first, followed by a description of the basic parameters to setup and solve the majority of optimization problems that NOMAD can handle. The library mode is described in *Optimization in library mode*.

NOMAD should be cited with references [AuCo04a] and [AuLeRoTr2021]. Other relevant papers by the developers are accessible through the NOMAD website http://www.gerad.ca/nomad.

References

7.1 Optimization in batch mode

The batch mode allows to separate the evaluation of the objectives and constraints by the blackbox program from NOMAD executable. This mode has the advantage that if your blackbox program crashes, it will not affect NOMAD: The point that caused this crash will simply be tagged as a blackbox failure.

Handling crashes in library mode requires special attention to isolate the part of code that may generate crashes. And, in general, using the library mode will require more computer programming than the batch mode. However, the library mode offers more options and flexibility for blackbox integration and management of optimization (see *Optimization in library mode*).

The different steps for solving your problem in batch mode are:

- Create a directory for your problem. The problem directory is where the NOMAD command is executed. It is a convenient place to put the blackbox executable, the parameters file and the output files, but those locations can be customized.
- Create your blackbox evaluation, which corresponds to a program (a binary executable or a script). This program can be located in the problem directory or not. This program outputs the objectives and the constraints for a given design vector. If you already have a blackbox program in a certain format, you need to interface it with a wrapper program to match NOMAD specifications (see *Getting started* for blackbox basics).

- Create a parameters file, for example `param.txt`. This file can be located in the problem directory or not (see *Basic parameters description* for more details).
- In the problem directory, start the optimization with a command like:

```
$NOMAD_HOME/bin/nomad param.txt
```

7.2 Basic parameters description

This section describes the basic parameters for the optimization problem definition, the algorithmic parameters and the parameters to manage output information. Additional information can be obtained by executing the command:

```
$NOMAD_HOME/bin/nomad -h
```

to see all parameters, or:

```
$NOMAD_HOME/bin/nomad -h PARAM_NAME
```

for a particular parameter.

The remaining content of a line is 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 `$` character is used in front of the names. For example, if a blackbox executable is run by the command `python script.py`, define parameter `BB_EXE "$python script.py"`.

Some parameters consists of a list of variable indices taken from 0 to $n-1$ (where n is the number of variables). Variable indices may be entered individually or as a range with format `i-j`. Character `*` may be used to replace 0 to $n-1$. 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. The strings `-`, `inf`, `-inf` or `+inf` are accepted to enter undefined real values (NOMAD considers $\pm\infty$ as an undefined value).

Parameters are classified into problem, algorithmic and output parameters, and provided in what follows. The advanced functionalities of NOMAD are presented in *Advanced functionalities*.

7.2.1 Problem parameters

Table 1: Basic problem parameters

Name	Argument	Short description	Default
BB_EXE	list of strings	blackbox executables (required in batch mode)	Empty string
BB_INPUT_TYPE	list of types	blackbox input types	`* R` (all real)
BB_OUTPUT_TYPE	list of types	blackbox output types (required)	`OBJ`
DIMENSION	integer	n the number of variables (required)	0
LOWER_BOUND	array of doubles	lower bounds	none
UPPER_BOUND	array of doubles	upper bounds	none

BB_EXE

In batch mode, BB_EXE indicates the names of the blackbox executables.

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.

A blackbox program can return more than one function *BB_OUTPUT_TYPE*:

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

A mapping between the names of the blackbox programs and the BB_OUTPUT_TYPE may be established to identify which function is returned by which blackbox:

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

Blackbox program names can be repeated to establish more complex mapping:

```
BB_EXE   bb1.exe bb2.exe bb2.exe   # defines TWO blackboxes
                                   # NO duplication if names are repeated
BB_OUTPUT_TYPE EB OBJ PB           # bb1.exe  has one output
                                   # bb2.exe  has two outputs
                                   # bb1.exe is executed first.
                                   #!! If EB constraint is feasible then
                                   #!!        bb2.exe is executed.
                                   #!! If EB constraint not feasible then
                                   #!!      bb2.exe is not launched.
```

A path can precede the blackbox program but spaces are not accepted in the path:

```
BB_EXE "dir_of_blackbox/bb.exe"
```

To prevent NOMAD from adding a path, the special character $ should be put in front of a command:

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

Or:

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

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 `` * t``. Input types `t` are values in `R, B, I`. `R` is for real/continuous variables, `B` for binary variables, and `I` for integer variables. The default type is `R`. See also *Detailed information*.

Note: Categorical variables are not yet supported in NOMAD 4 but are available in NOMAD 3.

BB_OUTPUT_TYPE

This parameter characterizes the values supplied by the blackbox, and in particular tells how constraint values are to be treated. 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. Currently, NOMAD 4 only supports single objective problem (NOMAD 3 can handle bi-objective). 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.

Note: A terminology is used to describe the different types of constraints [AuDe09a]

- `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` 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 [AuDe04a], and `PB` constraints are treated with the **Progressive Barrier** approach [AuDe09a].
- 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 corresponds by default to `PB` constraints.

All the types are:

`CNT_EVAL`	Must be 0 or 1: count or not the blackbox evaluation
`EB`	Constraint treated with **Extreme Barrier** (infeasible points are ignored)
`F`	Constraint treated with **Filter** approach
`NOTHING` `EXTRA_O` `-`	The output is ignored
`OBJ`	Objective value to be minimized
`PB` `CSTR`	Constraint treated with **Progressive Barrier**

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

LOWER_BOUND and UPPER_BOUND

> **Warning:** NOMAD is 0 based $\rightarrow$ The first variable has a 0 index.

Parameters LOWER_BOUND and UPPER_BOUND are used to define bounds on variables. For example, with $n = 7$:

```
LOWER_BOUND  0-2  -5.0
LOWER_BOUND  3     0.0
LOWER_BOUND  5-6  -4.0
UPPER_BOUND  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.
```

Each of these two sequences define the following bounds

$$
\begin{aligned}
-5 &\le x_0 \le 8 \\
-5 &\le x_1 \le 8 \\
-5 &\le x_2 \le 8 \\
0 &\le x_3 \le 8 \\
& x_4 \le 8 \\
-4 &\le x_5 \le 8 \\
-4 &\le x_6
\end{aligned}
$$

7.2.2 Algorithmic parameters

Table 2: Basic algorithmic parameters

Name	Argument	Short description	Default
DIRECTION_TYPE	direction type	type of directions for the poll	`ORTHO 2N`
F_TARGET	real t	NOMAD terminates if $f(x_k) \leq t$ for the objective function	none
INITIAL_MESH_SIZE	array of doubles	δ_0 [AuDe2006]	none
INITIAL_FRAME_SIZE	array of doubles	Δ_0 [AuDe2006]	`r0.1` or based on `X0`
LH_SEARCH	2 integers: `p0` and `pi`	**LH (Latin-Hypercube) search** (`p0`: initial and `pi`: iterative)	none
MAX_BB_EVAL	integer	maximum number of blackbox evaluations	none
MAX_TIME	integer	maximum wall-clock time (in seconds)	none
TMP_DIR	string	temporary directory for blackbox i/o files	problem directory
X0	point	starting point(s)	best point from a cache file or from an initial **LH search**

`DIRECTION_TYPE`

This parameter defines the type of directions for *Mads Poll* step. The possible arguments are:

Table 3: Direction types

`ORTHO 2N`	OrthoMADS, 2n. This corresponds to the original *Ortho Mads* algorithm [AbAuDeLe09] with $2n$ directions.
`ORTHO N+1 NEG`	OrthoMADS, n+1, with ((n+1)th dir = negative sum of the first n dirs) [AuIaLeDTr2014]
`N+1 UNI`	MADS with n+1, using $n+1$ uniformly distributed directions.
`SINGLE`	A single direction is produced
`DOUBLE`	Two opposite directions are produced

Multiple direction types may be chosen by specifying `DIRECTION_TYPE` several times.

`INITIAL_MESH_SIZE` and `INITIAL_FRAME_SIZE`

The *Poll* step initial frame size Δ_0 is decided by `INITIAL_FRAME_SIZE`. In order to achieve the scaling between variables, NOMAD considers the frame size parameter for each variable independently. The initial mesh size parameter `\delta_0` is decided based on `\Delta_0`. `INITIAL_FRAME_SIZE` may be entered with the following formats:

```
INITIAL_FRAME_SIZE     d0 (same initial mesh size for all variables)
INITIAL_FRAME_SIZE     (d0 d1 ... dn-1) (for all variables ``-`` may be used,  and␣
↪defaults will be considered)
```

(continues on next page)

(continued from previous page)

```
INITIAL_FRAME_SIZE i   d0 (initial mesh size provided for variable ``i`` only)
INITIAL_FRAME_SIZE i-j d0 (initial mesh size provided for variables ``i`` to ``j``)
```

The same logic and format apply for providing the INITIAL_MESH_SIZE, MIN_MESH_SIZE and MIN_FRAME_SIZE.

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/osxsystems, the directory /tmp is local and we advise the user to define TMP_DIR /tmp.

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).
- X0 keyword plus integer(s) 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
```

7.2.3 Output parameters

Table 4: Basic output parameters

Name	Argument	Short description	Default
CACHE_FILE	string	cache file; if the file does not exist, it will be created	none
DISPLAY_ALL_EVAL	bool	if `yes` all points are displayed with `DISPLAY_STATS` and `STATS_FILE`	`no`
DISPLAY_DEGREE	integer in [0 ; 3] or a string with four digits (see online help)	0 no display and 3 full display	`2`
DISPLAY_STATS	list of strings	what information is displayed at each success	`BBE OBJ`
HISTORY_FILE	string	file containing all trial points with format `x1 x2 ... xn bbo1 bbo2 ... bbom` on each line	none
SOLUTION_FILE	string	file to save the best feasible incumbent point in a simple format (SOL BBO)	none
STATS_FILE	string `file_name` + list of strings	the same as `DISPLAY_STATS` but for a display into file	none

DISPLAY_DEGREE

Four different levels of display can be set via the parameter `DISPLAY_DEGREE`. The `DISPLAY_MAX_STEP_LEVEL` can be used to control the number of steps displayed. To control the display of the **Models**, a `QUAD_MODEL_DISPLAY` and a `SGTELIB_MODEL_DISPLAY` are available. More information on these parameters can be obtained with online documentation: `$NOMAD_HOME/bin/nomad -h display`

DISPLAY_STATS and STATS_FILE

These parameters display information each time a new feasible incumbent (i.e. a new best solution) is found. `DISPLAY_STATS` is used to display at the standard output and `STATS_FILE` is used to write into a file. These parameters need a list of strings as argument, **without any quotes**. These strings may include the following keywords:

BBE	The number of blackbox evaluations
BBO	The blackbox outputs
OBJ	The objective function value
SOL	The current feasible iterate

Note: More display options are available. Check the online help: `$NOMAD_HOME/bin/nomad -h display_stats`

References

CHAPTER

EIGHT

OPTIMIZATION IN LIBRARY MODE

The library mode allows to tailor the evaluation of the objectives and constraints within a specialized executable that contains NOMAD shared object library.

For example, it is possible to link your own code with the NOMAD library (provided during installation) in a single executable that can define and run optimization for your problem. Contrary to the batch mode, this has the disadvantage that a crash within the executable (for example during the evaluation of a point) will end the optimization unless a special treatment of exception is provided by the user. But, as a counterpart, it offers more options and flexibility for blackbox integration and optimization management (display, pre- and post-processing, multiple optimizations, user search, etc.).

The library mode requires additional coding and compilation before conducting optimization. First, we will briefly review the compilation of source code to obtain NOMAD binaries (executable and shared object libraries) and how to use library. Then, details on how to interface your own code are presented.

8.1 Compilation of the source code

NOMAD source code files are located in `$NOMAD_HOME/src`. Examples are provided in `$NOMAD_HOME/examples/basic/library` and `$NOMAD_HOME/examples/advanced/library`.

The compilation procedure uses the provided `CMake` files along with the source code.

In what follows it is supposed that you have a write access to the source codes directory. If it is not the case, please consider making a copy in a more convenient location.

8.2 Using NOMAD libraries

Using the routines that are in the pre-compiled NOMAD shared object libraries (so) or dll (not yet available for Windows) with a `C++` program requires building an executable (*Installation* describes how to build the libraries and the examples). This is illustrated on the example located in the directory:

```
$NOMAD_HOME/examples/basic/library/example1
```

It is supposed that the environment variable `NOMAD_HOME` is defined and NOMAD shared object libraries are built. A basic knowledge of object oriented programming with `C++` is assumed. For this example, just one `C++` source file is used, but there could be a lot more.

8.2.1 Test the basic example 1

Let us first test the basic example to check that libraries are working fine and accessible. Library mode examples are built during the installation procedure (unless the flag `BUILD_LIBMODE_EXAMPLES` is set to `OFF`):

```
> cd $NOMAD_HOME/examples/basic/library/example1
> ls
CMakeLists.txt                example1_lib.cpp        example1_lib.exe
> ./example1_lib.exe
All variables are granular. MAX_EVAL is set to 1000000 to prevent algorithm from␣
↪circling around best solution indefinetely
BBE OBJ
1 -28247.525326  (Phase One)
12   -398.076167  (Phase One)
55   -413.531262
81  -1084.90725
136  -1632.176507
188  -1754.758402
201  -1787.5835
260  -1967.858372
764  -1967.860497
765  -1967.866871
766  -1967.885992
767  -1967.943355
768  -1968.115446
769  -1968.63171
770  -1970.180451
771  -1974.828012
772  -1988.763221
862  -1989.292074
863  -1990.878576
864  -1995.637558
895  -1999.023493
940  -1999.116474
942  -1999.395208
957  -1999.556452
961  -1999.835309
A termination criterion is reached: Max number of blackbox evaluations (Eval Global) No␣
↪more points to evaluate 1001

Best feasible solution:     #44485 ( 1.81678 5.21878 4.40965 8.1 15.1 8.8 5 10.1 1.6 5.5␣
↪)    Evaluation OK    f = -1999.835309000000052       h =   0

Best infeasible solution:   #44525 ( -26570.1 0 -5895.58 -3.58722e+06 -8.60934e+06 -5.
↪73956e+06 -1.36315e+07 5.73957e+06 1.14791e+07 2.86978e+06 )    Evaluation OK    f = -
↪1151.1639900000000125      h =   0.5625

Blackbox evaluations:        1001
Total model evaluations:     41241
Cache hits:                  104
Total number of evaluations: 1105
```

8.2.2 Modify CMake files

As a first task, you can create a CMakeLists.txt for your source code(s) based on the one for the basic example 1.

```
add_executable(example1_lib.exe example1_lib.cpp )
target_include_directories(example1_lib.exe PRIVATE ${CMAKE_SOURCE_DIR}/src)
set_target_properties(example1_lib.exe PROPERTIES INSTALL_RPATH "${CMAKE_INSTALL_PREFIX}/
↪lib")

if(OpenMP_CXX_FOUND)
  target_link_libraries(example1_lib.exe PUBLIC nomadAlgos nomadUtils nomadEval␣
↪OpenMP::OpenMP_CXX)
else()
  target_link_libraries(example1_lib.exe PUBLIC nomadAlgos nomadUtils nomadEval)
endif()

# installing executables and libraries
install(TARGETS example1_lib.exe  RUNTIME DESTINATION ${CMAKE_CURRENT_SOURCE_DIR} )

# Add a test for this example
if(BUILD_TESTS MATCHES ON)
   message(STATUS "    Add example library test 1")

   # Can run this test after install
   add_test(NAME Example1BasicLib COMMAND ${CMAKE_BINARY_DIR}/examples/runExampleTest.sh␣
↪./example1_lib.exe WORKING_DIRECTORY ${CMAKE_CURRENT_SOURCE_DIR} )
endif()
```

If you include your problem into the $NOMAD_HOME/examples directories, you just need to copy the example CMakeLists.txt into your own problem directory (for example $NOMAD_HOME/examples/basic/library/myPb), change the name example1_lib with your choice and add the subdirectory into $NOMAD_HOME/examples/CMakeLists.txt:

```
add_subdirectory(${CMAKE_CURRENT_SOURCE_DIR}/basic/library/myPb)
```

8.2.3 Modify C++ files

We now describe the other steps required for the creation of the source file (let us use example1.cpp) which is divided into two parts: a class for the description of the problem, and the main function.

The use of 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 classes NOMAD::Point or NOMAD::ArrayOfDouble. A lot of functionalities have been coded for theses classes, which are visible in files $NOMAD_HOME/src/Math/*.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.

Providing the blackbox evaluation of objective and constraints directly in the code avoids the use of temporary files and system calls by the algorithm. This is achieved by defining a derived class (let us call it My_Evaluator) that inherits from the class NOMAD::Evaluator. The blackbox evaluation is programmed in a user-defined class that will be automatically called by the algorithm.}

```
/**
 \file   example1_lib.cpp
 \brief  Library example for nomad
 \author Viviane Rochon Montplaisir
 \date   2017
 */

#include "Nomad/nomad.hpp"

/*----------------------------------------*/
/*               The problem              */
/*----------------------------------------*/
class My_Evaluator : public NOMAD::Evaluator
{
public:
    My_Evaluator(const std::shared_ptr<NOMAD::EvalParameters>& evalParams)
    : NOMAD::Evaluator(evalParams, NOMAD::EvalType::BB)
    {}

    ~My_Evaluator() {}

    bool eval_x(NOMAD::EvalPoint &x, const NOMAD::Double &hMax, bool &countEval) const␣
↪override
    {
        bool eval_ok = false;
        // Based on G2.
        NOMAD::Double f = 1e+20, g1 = 1e+20, g2 = 1e+20;
        NOMAD::Double sum1 = 0.0, sum2 = 0.0, sum3 = 0.0, prod1 = 1.0, prod2 = 1.0;
        size_t n = x.size();

        try
        {
            for (size_t i = 0; i < n ; i++)
            {
                sum1  += pow(cos(x[i].todouble()), 4);
                sum2  += x[i];
                sum3  += (i+1)*x[i]*x[i];
                prod1 *= pow(cos(x[i].todouble()), 2);
                if (prod2 != 0.0)
                {
                    if (x[i] == 0.0)
                    {
                        prod2 = 0.0;
                    }
                    else
                    {
                        prod2 *= x[i];
                    }
                }
            }

            g1 = -prod2 + 0.75;
            g2 = sum2 -7.5 * n;
```

(continues on next page)

(continued from previous page)

```
            f = 10*g1 + 10*g2;
            if (0.0 != sum3)
            {
                f -= ((sum1 -2*prod1) / sum3.sqrt()).abs();
            }
            // Scale
            if (f.isDefined())
            {
                f *= 1e-5;
            }

            NOMAD::Double c2000 = -f-2000;
            auto bbOutputType = _evalParams->getAttributeValue<NOMAD::BBOutputTypeList>(
↪"BB_OUTPUT_TYPE");
            std::string bbo = g1.tostring();
            bbo += " " + g2.tostring();
            bbo += " " + f.tostring();
            bbo += " " + c2000.tostring();

            x.setBBO(bbo);

            eval_ok = true;
        }
        catch (std::exception &e)
        {
            std::string err("Exception: ");
            err += e.what();
            throw std::logic_error(err);
        }

        countEval = true;
        return eval_ok;
    }
};
```

The argument `x` (in/out in `eval_x()`) 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 `x.setBBO(bbo);`. The outputs are returned as a string that will be interpreted by NOMAD based on the `BB_OUTPUT_TYPE` defined by the user. We recall that constraints must be represented by values c_j for a constraint $c_j \leq 0$.

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 [AuDe09a] for the definition of h and h_{max} and of the *Progressive Barrier* method for handling constraints.

The third argument, `countEval` (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 `countEval` to `false`).

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. 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 explicitely set the flag `countEval` to `true`.

8.2.4 Setting parameters

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.

```
/*------------------------------------------*/
/*            NOMAD main function           */
/*------------------------------------------*/
int main (int argc, char **argv)
{

    NOMAD::MainStep TheMainStep;

    auto params = std::make_shared<NOMAD::AllParameters>();
    initAllParams(params);
    TheMainStep.setAllParameters(params);

    std::unique_ptr<My_Evaluator> ev(new My_Evaluator(params->getEvalParams()));
    TheMainStep.setEvaluator(std::move(ev));

    try
    {
        TheMainStep.start();
        TheMainStep.run();
        TheMainStep.end();
    }

    catch(std::exception &e)
    {
        std::cerr << "\nNOMAD has been interrupted (" << e.what() << ")\n\n";
    }

    return 0;
}
```

The execution of NOMAD is controlled by the `NOMAD::MainStep` class using the `start`, `run` and `end` functions. The user defined `NOMAD::Evaluator` is set into the `NOMAD::MainStep`.

The base evaluator constructor takes an `NOMAD::EvalParameters` as input. The evaluation parameters are included into a `NOMAD::AllParameters`.

Hence, in library mode, the main function must declare a `NOMAD::AllParameters` object to set all types of parameters. Parameter names are the same as in batch mode but may be defined programmatically.

A parameter `PNAME` is set with the method `AllParameters::setAttributeValue( "PNAME", PNameValue)`. The `PNameValue` must be of a type registered for the `PNAME` parameter.

Warning: If the `PNameValue` has not the type associated to the `PName` parameters, the compilation will succeed but execution will be stopped when setting or getting the value.

Note: A brief description (including the `NOMAD::` type) of all parameters is given *Complete list of parameters*. More information on parameters can be obtained by running `$NOMAD_HOME/bin/nomad -h KEYWORD`.

For the example, the parameters are set in

```
void initAllParams(std::shared_ptr<NOMAD::AllParameters> allParams)
{
    // Parameters creation
    // Number of variables
    size_t n = 10;
    allParams->setAttributeValue( "DIMENSION", n);
    // The algorithm terminates after
    // this number of black-box evaluations
    allParams->setAttributeValue( "MAX_BB_EVAL", 1000);
    // Starting point
    allParams->setAttributeValue( "X0", NOMAD::Point(n, 7.0) );

    allParams->getPbParams()->setAttributeValue("GRANULARITY", NOMAD::ArrayOfDouble(n, 0.
↪0000001));

    // Constraints and objective
    NOMAD::BBOutputTypeList bbOutputTypes;
    bbOutputTypes.push_back(NOMAD::BBOutputType::PB);     // g1
    bbOutputTypes.push_back(NOMAD::BBOutputType::PB);     // g2
    bbOutputTypes.push_back(NOMAD::BBOutputType::OBJ);    // f
    bbOutputTypes.push_back(NOMAD::BBOutputType::EB);     // c2000
    allParams->setAttributeValue("BB_OUTPUT_TYPE", bbOutputTypes );

    allParams->setAttributeValue("DISPLAY_DEGREE", 2);
    allParams->setAttributeValue("DISPLAY_ALL_EVAL", false);
    allParams->setAttributeValue("DISPLAY_UNSUCCESSFUL", false);

    allParams->getRunParams()->setAttributeValue("HOT_RESTART_READ_FILES", false);
    allParams->getRunParams()->setAttributeValue("HOT_RESTART_WRITE_FILES", false);


    // Parameters validation
    allParams->checkAndComply();

}
```

The `checkAndComply` function must be called to ensure that parameters are compatible. Otherwise an exception is triggered.

8.2.5 Access to solution and optimization data

Not available yet

CHAPTER

NINE

PYNOMAD INTERFACE

Note: The Python interface requires Python 3.6 and Cython 0.24.

Note: Currently, PyNomad cannot be built when using Windows.

A Python interface for NOMAD is provided for Mac OS X and Linux. Some examples and source codes are provided in `$NOMAD_HOME/interfaces/PyNomad`. To enable the building of the Python interface, option `-DBUILD_INTERFACES=ON` must be set when configuring for building NOMAD, as such: `cmake -DBUILD_INTERFACES=ON -S . -B build/release`. The build procedure relies on Python 3.6 and Cython 0.24 or higher. A simple way to make it work is to first install the Anaconda package. The command `cmake --install build/release` must be run before using the PyNomad module.

All functionalities of NOMAD are available in PyNomad. NOMAD parameters are provided in a list of strings using the same syntax as used in the NOMAD parameter files. Several tests and examples are proposed in the `PyNomad` directory to check that everything is up and running.

CHAPTER

TEN

C INTERFACE

A C interface for NOMAD is provided for Mac OS X and Linux. The source codes are provided in `$NOMAD_HOME/interfaces/CInterface/`. To enable the building of the C interface, option `-DBUILD_INTERFACES=ON` must be set when building NOMAD, as such: `cmake -DBUILD_TESTS=ON -S . -B build/release`. The command `cmake --install build/release` must be run before using the library.

All functionalities of NOMAD are available in the C interface. NOMAD parameters are provided via these functions:

```
bool addNomadParam(NomadProblem nomad_problem, char *keyword_value_pair);
bool addNomadValParam(NomadProblem nomad_problem, char *keyword, int value);
bool addNomadBoolParam(NomadProblem nomad_problem, char *keyword, bool value);
bool addNomadStringParam(NomadProblem nomad_problem, char *keyword, char *param_str);
bool addNomadArrayOfDoubleParam(NomadProblem nomad_problem, char *keyword, double *array_
↪param);
```

See examples that are proposed in the `$NOMAD_HOME/examples/advanced/library/c_api` directory.

CHAPTER

ELEVEN

TRICKS OF THE TRADE

NOMAD has default values for all algorithmic parameters. These values represent a compromise between robustness and performance obtained by developers on sets of problems used for benchmarking. But you might want to improve NOMAD performance for your problem by tuning the parameters or use advanced functionalities. The following sections provide tricks that may work for you.

Here are a few suggestions for tuning NOMAD when facing different symptoms. The suggestions can be tested one by one or all together.

Table 1: Suggestions for tuning NOMAD

Symptom	Suggestion	Ref.
I want to see more display	Increase display degree	*DISPLAY_DEGREE*
Quantifiable constraints	Try PB EB or combinations	*BB_OUTPUT_TYPE*
Difficult constraint	Try PB instead of EB	*BB_OUTPUT_TYPE*
No initial point	Add a LH search	*LH Search and X0*
Variables of different magnitudes	Change blackbox input scaling	*Create blackbox programs*
	Change Δ_0 per variable	*INITIAL_MESH_SIZE and INITIAL_FRAME_SIZE*
	Tighten bounds	*LOWER_BOUND and UPPER_BOUND*
Many variables	Fix some variables	*FIXED_VARIABLE*
	Use *PSD-MADS*	*PSD-Mads*
Unsatisfactory solution	Change direction type to `N+1 UNI` or `N+1 NEG`	*DIRECTION_TYPE*
	Change initial point	*LH Search and X0*
	Add a LH search	*LH Search and X0*
	Add a VNS Mads search	*VNS Mads Search*
	Tighten bounds	*LOWER_BOUND and UPPER_BOUND*
	Change Δ_0	*INITIAL_MESH_SIZE and INITIAL_FRAME_SIZE*
	Modify seeds that affect algorithms	*SEED*
	Disable quadratic models	set `QUAD_MODEL_SEARCH no`
	Unable *SGTELIB* models	set `SGTELIB_MODEL_SEARCH yes`
	Disable opportunistic evaluations	set `EVAL_OPPORTUNISTIC no`
	Disable anisotropic mesh	set `ANISOTROPIC_MESH no`
	Change anisotropy factor	set `ANISOTROPY_FACTOR 0.05`
Improvements get negligible	Change stopping criteria	Type `nomad -h stop`
	Disable quadratic models	set `QUAD_MODEL_SEARCH no`
It takes long to improve f	Decrease Δ_0	*INITIAL_MESH_SIZE and INI-*

CHAPTER TWELVE

ADVANCED FUNCTIONALITIES

12.1 Advanced parameters

Advanced parameters are intended to setup optimization problems, algorithmic and output parameters when specific needs are present. Only a few advanced parameters are presented below; all advanced parameters can be obtained with `$NOMAD_HOME -h advanced`. Also a complete list of parameters and a short description is available in *Complete list of parameters*.

12.1.1 `EVAL_QUEUE_SORT`

Allows ordering of points before evaluation. This option has an effect only if the opportunistic strategy is enabled (parameter *EVAL_OPPORTUNISTIC*). The possible arguments are:

- `DIR_LAST_SUCCESS`: Points that are generated in a direction similar to the last direction that provided a successful point are evaluated first.
- `LEXICOGRAPHICAL`: Points are sorted in lexicographical order before evaluation.
- `RANDOM`: Mix points randomly before evaluation, instead of sorting them.
- `SURROGATE`: Sort points using values given by static surrogate. See parameter *SURROGATE_EXE*.

12.1.2 `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 vector of n values with format `(v0 v1 ... vn-1)`. Character `-` is used for free variables.
- An index range if at least one starting point has been defined. `FIXED_VARIABLE i-j`: variables `i` to `j` are fixed to their initial (`i-j` may be replaced by `i` only). See *X0* for practical examples of index ranges.

12.1.3 SEED

The directions that NOMAD explores during the *Poll* phase are dependent upon the seed. The seed is used to generate a pseudo-random direction on a unit n-dimensional sphere. The user can change the sequence of directions by setting `SEED` to a positive integer or `-1`. If `-1` or `DIFF` is entered the seed is different for each run (PID is used).

Other aspects of NOMAD may depend on a pseudo-random sequence of numbers depending on selected options: *LH Search* and *PSD Mads*.

12.1.4 EVAL_OPPORTUNISTIC

The opportunistic strategy consists in terminating the evaluations of a list of trial points at a given step of the algorithm as soon as an improved value is found.

This strategy is decided with the parameter `EVAL_OPPORTUNISTIC` and applies to both the *Poll* and *Search* steps. Search with NOMAD help `$NOMAD_HOME/bin/nomad -h OPPORTUNISTIC` for more options.

When evaluations are performed by blocks (see *Blackbox evaluation of a block of trial points*) the opportunistic strategy applies after evaluating a block of trial points.

12.1.5 VARIABLE_GROUP

By default NOMAD creates one group that combines all continuous, integer, and binary variables.

In batch mode, the `VARIABLE_GROUP` parameter followed by variable indices is used to explicitly form a group of variables. Each group of variable generates its own polling directions. The parameter may be entered several times to define more than one group of variables. Variables in a group may be of different types.

12.1.6 QUAD_MODEL_SEARCH and SGTELIB_MODEL_SEARCH

The *Search* phase of the *MADS* algorithm can use models of the objectives and constraints that are constructed dynamically from all the evaluations made. By default, a quadratic model is used to propose new points to be evaluated with the blackbox. To disable the use of quadratic models, the parameter `QUAD_MODEL_SEARCH` can be set to `no`.

Models from the *SGTELIB* library can be used by setting `SGTELIB_MODEL_SEARCH` to `yes`. Many parameters are available to control *SGTELIB* models: `$NOMAD_HOME/bin/nomad -h SGTELIB`.

12.1.7 VNS_MADS_SEARCH

The *Variable Neighborhood Search (VNS)* is a strategy to escape local minima.

The VNS Mads search strategy is described in [AuBeLe08b]. It is based on the Variable Neighborhood Search metaheuristic [MlHa97a] and [HaMl01a].

VNS Mads 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. Currently, the VNS Mads search will not use a surrogate if it is provided. This feature will be available in the future.

In NOMAD, the VNS Mads search strategy is not activated by default. In order to use the VNS Mads search, the user has to define the parameter `VNS_MADS_SEARCH`, with a boolean. The maximum desired ratio of VNS Mads blackbox evaluations over the total number of blackbox evaluations is specified with the real value parameter `VNS_MADS_SEARCH_TRIGGER`. For example, a value of 0.75 means that NOMAD will try to perform a maximum of 75% blackbox evaluations within the VNS Mads search. The default trigger ratio is 0.75.

12.1.8 GRANULARITY

The *MADS* algorithm handles granular variables, i.e. variables with a controlled number of decimals. For real numbers the granularity is 0. For integers and binary variables the granularity is automatically set to one.

The possible syntaxes to specify the granularity of the variables are as follows:

- n real values with format `GRANULARITY (v0 v1 ... vn-1)`.
- `GRANULARITY i-j v`: coordinates `i` to `j` set to `v`.
- `GRANULARITY * v`: all coordinates set to `v`.

12.1.9 SURROGATE_EXE

Static surrogate executable.

A static surrogate, or static surrogate function, is a cheaper blackbox function that is used, at least partially, to drive the optimization.

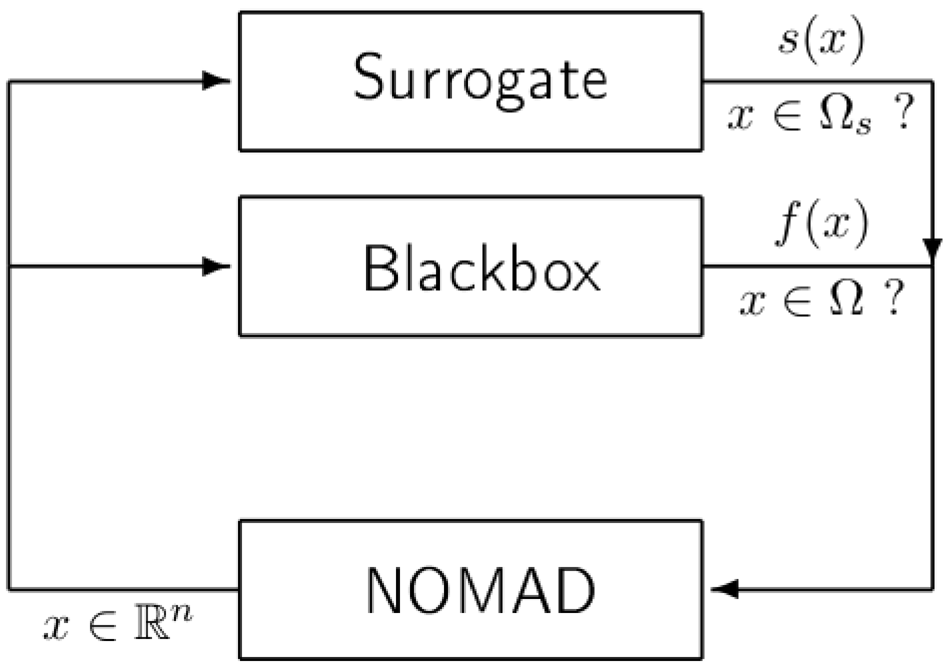


Fig. 1: Blackbox optimization using a surrogate

Note: The static surrogate is provided by the user.

The current version of NOMAD can use a static surrogate, provided by the user, which is not updated during the algorithm. See [BoDeFrSeToTr99a] for a survey on surrogate optimization, and [AuCM2019] about using static surrogate evaluations. This surrogate may be used for sorting points before evaluation (see parameter *EVAL_QUEUE_SORT*).

In batch mode, the parameter `SURROGATE_EXE` associates a static surrogate executable with the blackbox executable given by parameter `BB_EXE`. The surrogate must display the same input and output types as its associated blackbox, given by parameters `BB_INPUT_TYPE` and `BB_OUTPUT_TYPE`. In library mode, if a surrogate function is to be used, then its Evaluator should be of type `EvalType::SURROGATE` (see Section *Optimization in library mode*).

12.2 Blackbox evaluation of a block of trial points

At different phases of the MADS algorithm, different numbers of trial points are generated. For example, having selected the direction type as ORTHO 2N, the maximum number of points generated during the Poll step will be 2N+2. These points can be partitioned into blocks of trial points to be submitted sequentially for evaluation to a blackbox program. The maximum size of a block of evaluations is controlled by the BB_MAX_BLOCK_SIZE. By default, a block contains a single trial point. This can be changed by the user but the blackbox program must support the evaluation of a varying number of trial points, up to BB_MAX_BLOCK_SIZE.

Due to the strategy of by-block evaluation, the maximum number of evaluations requested to NOMAD may be exceeded if BB_MAX_BLOCK_SIZE > 1. The reason for this behaviour is that block results are analyzed only after completion and the maximum number of evaluations may be exceeded when checking this termination criterion. The opportunistic strategy (enabled by default) may apply after each block of trial points. Evaluations of blocks of trial points can be performed in parallel by the blackbox program. This strategy of parallelization must be setup by the user within the blackbox. Examples are provided in what follows.

12.2.1 Batch mode

In batch mode, NOMAD creates input files which can contain at most BB_MAX_BLOCK_SIZE trial points separated by a linebreak. Each point is given as a row of values. The user must provide a blackbox program that can read the input file, evaluate them and output the objective and constraints functions (in the order provided by the BB_OUTPUT_TYPE parameter) for each trial point in the same order as provided in the input file. A blackbox program may fail to evaluate some of the trial points. When block of trial points is submitted the content of the output file must reflect the outputs for each point. If one value provided in the output file cannot be read by NOMAD, then the corresponding trial point is considered as having failed. The trial points that have failed will not be evaluated again. An example of blackbox program written is provided in the directory `$NOMAD_HOME/examples/basic/batch/single_obj_parallel`. The executable `bb3.exe` evaluates up to 4 trial points in parallel.

```
> cd $NOMAD_HOME/examples/basic/batch/single_obj_parallel
> more x.txt
1 2 3 4 5
0 0 0 0 0
2 2 2 2 2
5 4 3 2 1
> bb3.exe x.txt
5 5 -65
0 -20 20
2 -20 -20
1 5 -65
```

The same directory holds the parameter file that specifies this blackbox program with blocks of 4 trial points:

```
DIMENSION      5              # number of variables

BB_EXE bb3.exe
BB_MAX_BLOCK_SIZE 4

BB_OUTPUT_TYPE OBJ PB EB

```

(continues on next page)

(continued from previous page)

```
X0              ( 0 0 0 0 0 )  # starting point

LOWER_BOUND     * -6.0         # 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_BLOCK_EVAL     20          # the algorithm terminates when
                               # 20 blocks have been evaluated

TMP_DIR /tmp
DISPLAY_DEGREE 2
DISPLAY_STATS BLK_EVA BLK_SIZE OBJ
DISPLAY_ALL_EVAL true
```

When evaluations are performed by blocks, i.e., when `BB_MAX_BLOCK_SIZE` is greater than one, the opportunistic strategy applies after evaluating a block of trial points.

12.2.2 Library mode

Please refer to `$NOMAD_HOME/examples/basic/library/single_obj_parallel` for an example on how to manage a block of evaluations in parallel using OpenMP.

12.3 Parallel evaluations

When OpenMP is available (see *Use OpenMP*), the user may provide the number of threads `NB_THREADS_OPENMP` to efficiently access the computer cores. If this parameter is not set, OpenMP computes the number of available threads. The evaluations of trial points are dispatched to these threads.

12.4 PSD-Mads

The PSD-MADS method implements a parallel space decomposition of MADS and is described in [AuDeLe07]. The method aims at solving larger problems than the scalar version of NOMAD. NOMAD is in general efficient for problems with up to about 20 variables, PSD-MADS has solved problems with up to 500 variables. In PSD-MADS, each worker process has the responsibility for a small number of variables on which a MADS algorithm is performed. These subproblems are decided by the PSD-MADS algorithm. These groups of variables are chosen randomly, without any specific strategy. A special worker, called the pollster, works on all the variables, but with a reduced number of directions. The pollster ensures the convergence of the algorithm. Concerning other aspects, the algorithm given here is similar to the program PSD-MADS given with NOMAD 3.

The management of parallel processes is done using OpenMP. To use PSD-MADS, set parameter `PSD_MADS_OPTIMIZATION` to `true`. Thread 0 is used for the pollster. The next `PSD_MADS_NB_SUBPROBLEM` threads are used for subproblems. If this parameter is not set, it is computed using `PSD_MADS_NB_VAR_IN_SUBPROBLEM`. Remaining available threads are not used for algorithmic management or point generation, only for point evaluation. An example of usage of PSD-MADS in library mode is in `$NOMAD_HOME/examples/advanced/library/PSDMads`.

12.5 Hot and Warm Restart

This new feature of NOMAD 4 makes it possible to continue the solving process after it has started, without having to restart it from the beginning. In the case of hot restart, the user interrupts the solver to change the value of a parameter. With warm restart, the user changes a parameter from a resolution that has already reached a termination condition. In both cases, the solving process is then continued from its current state.

12.5.1 Hot restart

To enable hot restart, set parameter `HOT_RESTART_ON_USER_INTERRUPT` to `true`. While NOMAD is running, interrupt the run with the command `CTRL-C`. New values for parameters may be entered. For example, entering `LH_SEARCH 0 20` will make LH search be used for the rest of the optimization. The syntax is the same as the syntax of a parameter file, when in batch mode. When all new parameter values are entered, continue optimization by entering the command `CTRL-D`. The new parameter values will be taken into account.

12.5.2 Warm restart

To enable warm restart, parameters `HOT_RESTART_READ_FILES` and `HOT_RESTART_WRITE_FILES` need to be set to `true`. When NOMAD runs a first time, files `hotrestart.txt` and `cache.txt` are written to the problem directory. This information is used if NOMAD is run a second time. Instead of redoing the same optimization, NOMAD will continue where it was when the first run was ended. For example, suppose the first NOMAD run stopped at evaluation 100 because the value of parameter `MAX_BB_EVAL` was 100. The user still has room for 50 more evaluations. The parameter file may be changed with value `MAX_BB_EVAL 150`, and the second run of NOMAD will start where it was, with evaluation 101.

12.6 Doxygen

A local doxygen documentation can be created by running the `doxygen` command (if available) in `$NOMAD_HOME/doc/doxygen`. The documentation can be opened by a browser at `$NOMAD_HOME/doc/doxygen/html/index.html`.

References

CHAPTER

THIRTEEN

RELEASE NOTES AND FUTURE DEVELOPMENTS

NOMAD 4 is a complete redesign compared with NOMAD 3, with a new architecture providing more flexible code, some added functionalities and reusable code.

Some functionalities available in NOMAD 3 will be included in NOMAD 4 in future releases:

- *BiMads* [AuSaZg2008a]
- *RobustMads* [AudIhaLedTrib2016] and *StoMads* [G-2019-30]
- Categorical [AuDe01a] and periodical variables [AuLe2012]

The performance of NOMAD 4 and 3 are similar when the default parameters of NOMAD 4 are used (see [AuLeRoTr2021]).

References

CHAPTER

FOURTEEN

COMPLETE LIST OF PARAMETERS

A set of parameters is available in the table below for fine tuning algorithmic settings. Additional information on each parameter is available by typing `$NOMAD_HOME/bin/nomad -h PARAM_NAME`.

Table 1: NOMAD 4 parameters

Name	Type	Argument	Short description	Default
ADD_SEED_TO_FILE_NAMES	bool	advanced	The flag to add seed to the file names	true
ANISOTROPIC_MESH	bool	advanced	MADS uses anisotropic mesh for generating directions	true
ANISOTROPY_FACTOR	NOMAD::Double	advanced	MADS anisotropy factor for mesh size change	0.1
BB_EXE	std::string	basic	Blackbox executable	No default
BB_INPUT_TYPE	NOMAD::BBInputTypeList	basic	The variable blackbox input types	* R
BB_MAX_BLOCK_SIZE	size_t	advanced	Size of blocks of points, to be used for parallel evaluations	1
BB_OUTPUT_TYPE	NOMAD::BBOutputTypeList	basic	Type of outputs provided by the blackboxes	OBJ
CACHE_FILE	std::string	basic	Cache file name	No default
CACHE_SIZE_MAX	size_t	advanced	Maximum number of evaluation points to be stored in the cache	INF
DIMENSION	size_t	basic	Dimension of the optimization problem (required)	0
DIRECTION_TYPE	NOMAD::DirectionTypeList	advanced	Direction types for Mads Poll step	ORTHO 2N
DIRECTION_TYPE_SECONDARY_POLL	NOMAD::DirectionTypeList	advanced	Direction types for Mads secondary poll	DOUBLE
DISPLAY_ALL_EVAL	bool	basic	Flag to display all evaluations	false
DISPLAY_DEGREE	int	basic	Level of verbose during execution	2
DISPLAY_HEADER	size_t	advanced	Frequency at which the stats header is displayed	40
DISPLAY_INFEASIBLE	bool	advanced	Flag to display infeasible	false
DISPLAY_MAX_STEP_LEVEL	size_t	advanced	Depth of the step after which info is not printed	20
DISPLAY_STATS	NOMAD::ArrayOfString	basic	Format for displaying the evaluation points	BBE OBJ
DISPLAY_UNSUCCESSFUL	bool	advanced	Flag to display unsuccessful	true

continues on next page

Table 1 – continued from previous page

EVAL_OPPORTUNISTIC	bool	advanced	Opportunistic strategy: Terminate evaluations as soon as a success is found	true
EVAL_QUEUE_CLEAR	bool	advanced	Opportunistic strategy: Flag to clear EvaluatorControl queue between each run	true
EVAL_QUEUE_SORT	NOMAD::EvalSortType	advanced	How to sort points before evaluation	DIR_LAST_SUCCESS
EVAL_STATS_FILE	string	basic	The name of the file for stats about evaluations and successes	No default
EVAL_SURROGATE_COST	size_t	advanced	Cost of the surrogate function versus the true function	INF
EVAL_SURROGATE_OPTIMIZATION	bool	advanced	Use static surrogate as blackbox for optimization	false
EVAL_USE_CACHE	bool	advanced	Use cache in algorithms	true
FIXED_VARIABLE	NOMAD::Point	advanced	Fix some variables to some specific values	No default
FRAME_CENTER_USE_CACHE	bool	advanced	Find best points in the cache and use them as frame centers	false
GRANULARITY	NOMAD::ArrayOfDouble	advanced	The granularity of the variables	No default
HISTORY_FILE	std::string	basic	The name of the history file	No default
H_MAX_0	NOMAD::Double	advanced	Initial value of hMax.	NOMAD::INF
HOT_RESTART_FILE	std::string	advanced	The name of the hot restart file	hotrestart.txt
HOT_RESTART_ON_USER_INTERRUPT	bool	advanced	Flag to perform a hot restart on user interrupt	false
HOT_RESTART_READ_FILES	bool	advanced	Flag to read hot restart files	false
HOT_RESTART_WRITE_FILES	bool	advanced	Flag to write hot restart files	false
INITIAL_FRAME_SIZE	NOMAD::ArrayOfDouble	advanced	The initial frame size of MADS	No default
INITIAL_MESH_SIZE	NOMAD::ArrayOfDouble	advanced	The initial mesh size of MADS	No default
LH_EVAL	size_t	basic	Latin Hypercube Sampling of points (no optimization)	0
LH_SEARCH	NOMAD::LHSearchType	basic	Latin Hypercube Sampling Search method	No default
LOWER_BOUND	NOMAD::ArrayOfDouble	basic	The optimization problem lower bounds for each variable	No default
MAX_BB_EVAL	size_t	basic	Stopping criterion on the number of blackbox evaluations	INF
MAX_BLOCK_EVAL	size_t	basic	Stopping criterion on the number of blocks evaluations	INF
MAX_EVAL	size_t	advanced	Stopping criterion on the number of evaluations (blackbox and cache)	INF
MAX_ITERATION_PER_MEGAITERATION	size_t	advanced	Maximum number of Iterations to generate for each MegaIteration.	INF
MAX_ITERATIONS	size_t	advanced	The maximum number of iterations of the MADS algorithm	INF
MAX_SURROGATE_EVAL_OPTIMIZATION	size_t	basic	Stopping criterion on the number of static surrogate evaluations	INF
MAX_TIME	size_t	basic	Maximum wall-clock time in seconds	INF
MEGA_SEARCH_POLL	bool	advanced	Evaluate points generated from Search and Poll steps all at once	false
MIN_FRAME_SIZE	NOMAD::ArrayOfDouble	basic	Termination criterion on minimal frame size of MADS	No default
MIN_MESH_SIZE	NOMAD::ArrayOfDouble	basic	Termination criterion on minimal mesh size of MADS	No default
NB_THREADS_OPENMP	int	advanced	The number of threads when OpenMP parallel evaluations	-1

continues on next page

Table 1 – continued from previous page

NM_DELTA_E	NOMAD::Double	advanced	NM expansion parameter delta_e.	2
NM_DELTA_IC	NOMAD::Double	advanced	NM inside contraction parameter delta_ic.	-0.5
NM_DELTA_OC	NOMAD::Double	advanced	NM outside contraction parameter delta_oc.	0.5
NM_GAMMA	NOMAD::Double	advanced	NM shrink parameter gamma.	0.5
NM_OPTIMIZATION	bool	advanced	Nelder Mead stand alone optimization for constrained and unconstrained pbs	false
NM_SEARCH	bool	advanced	Nelder Mead optimization used as a search step for Mads	true
NM_SEARCH_MAX_TRIAL_PTS_NFACTOR	size_t	advanced	NM-Mads search stopping criterion.	80
NM_SEARCH_RANK_EPS	NOMAD::Double	advanced	NM-Mads epsilon for the rank of DZ.	0.01
NM_SEARCH_STOP_ON_SUCCESS	bool	advanced	NM-Mads search stops on success.	false
NM_SIMPLEX_INCLUDE_FACTOR	size_t	advanced	Construct NM simplex using points in cache.	8
NM_SIMPLEX_INCLUDE_LENGTH	NOMAD::Double	advanced	Construct NM simplex using points in cache.	INF
PSD_MADS_ITER_OPPORTUNISTIC	bool	advanced	Opportunistic strategy between the Mads subproblems in PSD-MADS	true
PSD_MADS_NB_SUBPROBLEM	size_t	advanced	Number of PSD-MADS subproblems	INF
PSD_MADS_NB_VAR_IN_SUBPROBLEM	size_t	advanced	Number of variables in PSD-MADS subproblems	2
PSD_MADS_OPTIMIZATION	bool	advanced	PSD-MADS optimization algorithm	0
PSD_MADS_ORIGINAL	bool	advanced	Use NOMAD 3 strategy for mesh update in PSD-MADS	false
PSD_MADS_SUBPROBLEM_MAX_BB_EVAL	size_t	advanced	Max number of evaluations for each subproblem	INF
PSD_MADS_SUBPROBLEM_PERCENT_COVERAGE	NOMAD::Double	advanced	Percentage of variables that must be covered in subproblems before updating mesh	70
QUAD_MODEL_DISPLAY	std::string	developer	Display of a model	No default
QUAD_MODEL_MAX_BLOCK_SIZE	size_t	advanced	Size of blocks of points, to be used for parallel evaluations	INF
QUAD_MODEL_MAX_EVAL	size_t	advanced	Max number of model evaluations for each optimization of the quad model problem	1000
QUAD_MODEL_OPTIMIZATION	bool	advanced	Quad model stand alone optimization for constrained and unconstrained pbs	false
QUAD_MODEL_RADIUS_FACTOR	NOMAD::Double	developer	Quadratic model radius factor	2.0
QUAD_MODEL_SEARCH	bool	basic	Quad model search	true
QUAD_MODEL_SEARCH_BOUND_REDUCTION_FACTOR	NOMAD::Double	basic	Scale the bounds for the quad model search	1
REJECT_UNKNOWN_PARAMETERS	bool	advanced	Flag to reject unknown parameters when checking validity of parameters	false
RHO	NOMAD::Double	advanced	Rho parameter of the progressive barrier	0.1
SEED	int	advanced	The seed for the pseudo-random number generator	0
SGTELIB_MAX_POINTS_FOR_MODEL	size_t	developer	Maximum number of valid points used to build a model	100
SGTELIB_MIN_POINTS_FOR_MODEL	size_t	developer	Minimum number of valid points necessary to build a model	1
SGTELIB_MODEL_DEFINITION	NOMAD::ArrayOfString	advanced	Definition of the Sgtelib model	No default

continues on next page

Table 1 – continued from previous page

SGTELIB_MODEL_DISPLAY	std::string	developer	Display of a model	No default
SGTELIB_MODEL_DIVERSIFICATION	NOMAD::Double	developer	Coefficient of the exploration term in the sgtelib model problem	0.01
SGTELIB_MODEL_EVAL	bool	advanced	Sgtelib Model Sampling of points	0
SGTELIB_MODEL_FEASIBILITY	NOMAD::SgtelibModelFeasibilityType	developer	Method used to model the feasibility of a point	C
SGTELIB_MODEL_FORMULATION	NOMAD::SgtelibModelFormulationType	developer	Formulation of the sgtelib model problem	FS
SGTELIB_MODEL_MAX_BLOCK_SIZE	size_t	advanced	Size of blocks of points, to be used for parallel evaluations	INF
SGTELIB_MODEL_MAX_EVAL	size_t	advanced	Max number of model evaluations for each optimization of the sgtelib model problem	1000
SGTELIB_MODEL_RADIUS_FACTOR	NOMAD::Double	developer	Sgtelib model radius factor	2.0
SGTELIB_MODEL_SEARCH	bool	basic	Model search using Sgtelib	false
SGTELIB_MODEL_SEARCH_CANDIDATES_NB	int	developer	Number of candidates returned by the sgtelib model search	-1
SGTELIB_MODEL_SEARCH_EXCLUSION_AREA	NOMAD::Double	developer	Exclusion area for the sgtelib model search around points of the cache	0.0
SGTELIB_MODEL_SEARCH_FILTER	std::string	developer	Methods used in the sgtelib search filter to return several search candidates	2345
SGTELIB_MODEL_SEARCH_TRIALS	size_t	developer	Max number of sgtelib model search failures before going to the poll step	1
SOLUTION_FILE	std::string	basic	The name of the file containing the best feasible solution	No default
SPECULATIVE_SEARCH_BASE_FACTOR	NOMAD::Double	advanced	Distance of the MADS speculative search method	4.0
SPECULATIVE_SEARCH	bool	basic	MADS speculative search method	true
SPECULATIVE_SEARCH_MAX	size_t	advanced	MADS speculative search method	1
SSD_MADS_ITER_OPPORTUNISTIC	bool	advanced	Opportunistic strategy between the Mads subproblems in SSD-MADS	true
SSD_MADS_NB_SUBPROBLEM	size_t	advanced	Number of SSD-MADS subproblems	INF
SSD_MADS_NB_VAR_IN_SUBPROBLEM	size_t	advanced	Number of variables in SSD-MADS subproblems	2
SSD_MADS_OPTIMIZATION	bool	advanced	SSD-MADS optimization algorithm	0
SSD_MADS_RESET_VAR_PICKUP_SUBPROBLEM	bool	advanced	Reset random variable pick-up for each subproblem	false
SSD_MADS_SUBPROBLEM_MAX_BB_EVAL	size_t	advanced	Max number of evaluations for each subproblem	INF
STATS_FILE	NOMAD::ArrayOfString	basic	The name of the stats file	No default
STOP_IF_FEASIBLE	bool	advanced	Stop algorithm once a feasible point is obtained	false
STOP_IF_PHASE_ONE_SOLUTION	bool	advanced	Stop algorithm once a phase one solution is obtained	false
SURROGATE_EXE	std::string	advanced	Static surrogate executable	No default
TMP_DIR	std::string	advanced	Directory where to put temporary files	No default
UPPER_BOUND	NOMAD::ArrayOfDouble	basic	The optimization problem upper bounds for each variable	No default
USER_CALLS_ENABLED	bool	advanced	Controls the automatic calls to user function	true
VARIABLE_GROUP	NOMAD::ListOfVariableGroup	advanced	The groups of variables)	No default

continues on next page

Table 1 – continued from previous page

VNS_MADS_OPTIMIZATION	bool	advanced	VNS MADS stand alone optimization for constrained and unconstrained pbs	false
VNS_MADS_SEARCH	bool	advanced	VNS Mads optimization used as a search step for Mads	false
VNS_MADS_SEARCH_MAX_TRIAL_PTS_NFACTOR	size_t	advanced	VNS-Mads search stopping criterion.	100
VNS_MADS_SEARCH_TRIGGER	NOMAD::Double	advanced	VNS Mads search trigger	0.75
X0	NOMAD::ArrayOfPoint	basic	The initial point(s)	No default

14.1 Detailed information

In progress

BB_INPUT_TYPE

```
Type: NOMAD::BBInputTypeList

Default: * R

Description:

. Blackbox input types

. List of types for each variable

. Available types:
  . B: binary
  . I: integer
  . R: continuous

. Examples:
  . BB_INPUT_TYPE * I       # all variables are integers
  . BB_INPUT_TYPE ( R I B ) # for all 3 variables
  . BB_INPUT_TYPE 1-3 B     # NOT YET SUPPORTED ( variables 1 to 3 are binary )
  . BB_INPUT_TYPE 0 I       # NOT YET SUPPORTED ( first variable is integer )
```

DIMENSION

```
Type: size_t

Default: 0

Description :

. Number of variables

. Argument: one positive integer

. Example: DIMENSION 3
```

CHAPTER

FIFTEEN

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[AuLeRoTr2021] C. Audet, S. Le Digabel, V. Rochon Montplaisir, and C. Tribes. NOMAD version 4: Nonlinear optimization with the MADS algorithm. Submitted.

[AuHa2017] C. Audet and W. Hare. Derivative-Free and Blackbox Optimization. *Springer Series in Operations Research and Financial Engineering*. Springer International Publishing, Berlin, 2017.

[AbAuDeLe09] 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.

[AuDe2006] C. Audet and J.E. Dennis, Jr. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization*, 17(1):188–217, 2006.

[AuDe09a] C. Audet and J.E. Dennis, Jr. A Progressive Barrier for Derivative-Free Nonlinear Programming. *SIAM Journal on Optimization*, 20(1):445–472, 2009.

[AuCo04a] M.A. Abramson, C. Audet, G. Couture, J.E. Dennis, Jr., S. Le Digabel, V. Rochon Montplaisir, and C. Tribes. The NOMAD project. Software available at https://www.gerad.ca/nomad, 2021.

[AuDe04a] 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.

[AuIaLeDTr2014] C. Audet and A. Ianni and S. Le~Digabel and C. Tribes. Reducing the Number of Function Evaluations in Mesh Adaptive Direct Search Algorithms. *SIAM Journal on Optimization*, 24(2):621-642, 2014.

[AuBeLe08b] C. Audet, V. Béchard, and S. Le Digabel. Nonsmooth optimization through mesh adaptive direct search and variable neighborhood search. *Journal of Global Optimization*, 41(2):299– 318, 2008.

[AuCM2019] C. Audet and J. Côté-Massicotte. Dynamic improvements of static surrogates in direct search optimization. *Optimization Letters* 13, 6 (2019), 1433-1447

[AuDeLe07] 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.

[BoDeFrSeToTr99a] 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.

[HaMl01a] P. Hansen and N. Mladenović. Variable neighborhood search: principles and applications. *European Journal of Operational Research*, 130(3):449–467, 2001.

[MlHa97a] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers and Operations Research*, 24(11):1097–1100, 1997.

[AuSaZg2008a] C. Audet, G. Savard, and W. Zghal. 2008. Multiobjective Optimization Through a Series of Single-Objective Formulations. *SIAM Journal onOptimization* 19, 1 (2008), 188–210

[AudIhaLedTrib2016] C. Audet, A. Ihaddadene, S. Le Digabel, and C. Tribes. 2018. Robust optimization of noisy blackbox problems using the Mesh Adaptive Direct Search algorithm. *Optimization Letters* 12, 4 (2018), 675–689

[G-2019-30] C. Audet, K.J. Dzahini, M. Kokkolaras, and S. Le Digabel. 2021.Stochastic mesh adaptive direct search for blackbox optimization using probabilistic estimates. *Technical Report* G-2019-30. Les cahiers du GERAD. To appear in *Computational Optimization and Applications*.

[AuDe01a] C. Audet and J.E. Dennis, Jr. 2001. Pattern Search Algorithms for Mixed Variable Programming. *SIAM Journal on Optimization* 11, 3 (2001), 573–594.

[AuLe2012] C. Audet and S. Le Digabel. 2012. The mesh adaptive direct search algorithm for periodic variables. *Pacific Journal of Optimization* 8, 1 (2012),103–119