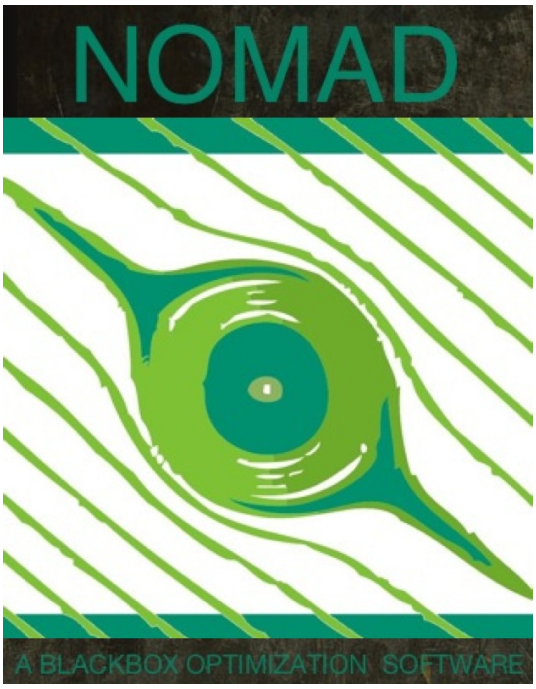


# NOMAD User Guide
# Version 3.6.1

*Sébastien Le Digabel, Christophe Tribes and Charles Audet*

**How to use this guide:**

- A general introduction of NOMAD is presented in Chapter 1.
- New users of NOMAD:
  Chapter 2 describes how to install the software application.
  Chapter 3 describes how to get started with NOMAD.
- NOMAD utilization:
  All users can find in Chapters 4 to 7 ways to tailor problem definition, algorithmic settings and software output.
- Tricks that may help solving specific problems are presented in Chapter 5.

**Please cite NOMAD with references [25, 50].**

# List of acronyms

| | |
|---|---|
| NOMAD | Nonlinear Optimization by Mesh Adaptive Direct Search (software) |
| MADS | Mesh Adaptive Direct Search (algorithm) |
| LT-MADS | Original MADS |
| OrthoMADS | Second MADS |
| BiMADS | Biobjective MADS algorithm |
| p-MADS | Parallel version of MADS |
| Coop-MADS | Parallel version of MADS with cooperation |
| PSD-MADS | Parallel version of MADS with space decomposition |
| VNS | Variable Neighborhood Search |
| GPS | Generalized Pattern Search |

# Contents

# Preface

In many situations, one is interested in identifying the values of a set of variables that maximize or minimize some objective function. Furthermore, the variables cannot take arbitrary values, as they are confined to an admissible region and need to satisfy some prescribed requirements. NOMAD is a software application designed to solve these kind of problems.

The nature of the objective function and constraints dictates the type of optimization methods that should be used to tackle a given problem. If the optimization problem is convex, or if the functions are smooth and easy to evaluate, or if the number of variables is large, then NOMAD is not the solution that you should use. NOMAD is intended for time-consuming blackbox simulation with a small number of variables. NOMAD is often useful when other optimizers fail.

These nasty problems are called *blackbox optimization problems*. With NOMAD some constraints may be evaluated prior to launching the simulation, and others may only be evaluated *a posteriori*. The simulations may take several seconds, minutes hours or even days to compute. The blackbox can have limited precision and be contaminated with numerical noise. It may also fail to return a valid output, even when the input appears acceptable. Launching twice the simulation from the same input may produce different outputs. These unreliable properties are frequently encountered when dealing with real problems. The term blackbox is used to indicate that the internal structure of the target problem, such as derivatives or their approximations, cannot be exploited as it may be unknown, hidden, unreliable or inexistent. There are situations where some structure such as bounds may be exploited and in some cases, a surrogate of the problem may be supplied to NOMAD or a model may be constructed and trusted.

This document describes how to use NOMAD to solve your blackbox optimization problem.

# Part I

# FIRST NOMAD STEPS

# Chapter 1

# Introduction

## 1.1 What is NOMAD?

NOMAD = **N**onlinear **O**ptimization by **M**esh **A**daptive **D**irect 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 (see Figure 1.1). 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 [8, 18, 20] designed for constrained optimization of blackbox functions in the form

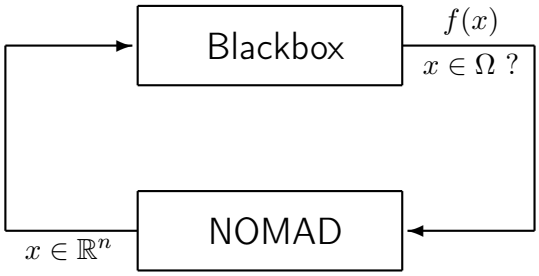


Figure 1.1: NOMAD blackbox optimization.

$$\min_{x \in \Omega} f(x) \tag{1.1}$$

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 $\mathbb{R}^n$.

## 1.2 Basics of the MADS algorithm

At the core of NOMAD resides the Mesh Adaptive Direct Search (MADS) algorithm. As the name implies, these methods generate iterates on a tower of underlying meshes on the domain space. A mesh is a discretization of the space of variables. However, also as the name implies, they perform an adaptive search on the meshes including controlling the refinement of the meshes. The reader interested in the rather technical details should read [18].

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. The SEARCH step is crucial in practice because it is so flexible, but it is a difficulty for the theory for the same reason. SEARCH can return any point on the underlying mesh, but of course, it is trying to identify 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*;

In addition NOMAD includes the following algorithms:

- A MIXED VARIABLE PROGRAMMING (MVP) algorithm to optimize with respect to mixtures of discrete, continuous, and categorical decision variables (see Section 7.1).
- A BIMADS (BIOBJECTIVE MESH ADAPTIVE DIRECT SEARCH) algorithm to consider a biobjective version of (1.1) (see Section 7.2).
- A VARIABLE NEIGHBORHOOD SEARCH (VNS) algorithm (see Section 7.5) to escape local minima.
- Three algorithms for parallel executions (see Section 7.3).

## 1.3 Using NOMAD


NOMAD has no graphical user interface

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 Chapter 2. New users should refer to Chapter 3 to get started. The most important instructions to use NOMAD are in Chapter 4. In addition, tricks that may help solving specific problems and improve NOMAD efficiency are presented in Chapter 5. Advanced parameters and functionalities are presented in Chapters 6 and 7.

## 1.4 Licence

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 licence obtained during installation. For additional information you can contact us or visit the Free Software Foundation web site.

## 1.5 Contact us

Contact information:
École Polytechnique de Montréal - GERAD
C.P. 6079, Succ. Centre-ville, Montréal (Québec) H3C 3A7 Canada
e-mail: nomad@gerad.ca
fax : 1-514-340-5665

All queries can be submitted by email at `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).

Support and bug report: `nomad@gerad.ca`

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

## 1.6 Supported plateforms and environments

NOMAD source codes are in C++ and are identical for all supported plateforms.

For convenience, the NOMAD installation packages are customized depending on the plateform. The *Mac OS X* and *Windows* installation packages contain executables to quickly start using NOMAD without having to compile the sources. The *Mac OS X* version of the executable is compiled with gcc (g++), version 4. The *Windows* version of the executable is compiled with *Visual Studio C++ 2010*.

The *Linux* installation package contains no executable but the source codes include a standard makefile for compilation. The compilation has been tested with gcc (g++), version 4.

NOMAD supports parallel evaluations of blackboxes. However, this capability is obtained by compiling a parallel version of the source codes using the message passing interface (*MPI* [58]). Details on how to proceed are provided in Section 7.3.

A *Matlab* version for *Windows* can be obtained at OPTI Toolbox website.

**Tested operating systems and environments:**

- *Unix*, *Linux* & *Mac OS X*
- *Windows* Xp and *Windows* 7
- *Matlab 2010a* or above for *Windows* x86 (32bit) and *Windows* x64 (64bit)

The installation procedure is presented in Chapter 2.

## 1.7 Authors and fundings

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

Developers of the methods 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/Sebastien.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 3.5.1 (and above) of NOMAD is developed by Christophe Tribes. Version 3.0 (and above) was developed by Sébastien Le Digabel. Previous versions were written by Gilles Couture (GERAD).

## 1.8 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 and Klaus Truemper for their feedbacks and tests that significantly contributed to improve NOMAD.

Finally, many thanks to the TOMS anonymous referees for their useful comments which helped a lot to improve the code and the text of [50].

## 1.9 Type conventions

The following conventions are used in this document:

- *Software and operating systems names are typeset in this font.*
- NOMAD is typeset in uppercase.
- `Program codes, program executables, shell commands and environment variables are typeset in this font.`
- `Parameter names are typeset in this font using uppercase.`
- Algorithm names are typeset in this font.

- Important information is highlighted in a box like this.

# Chapter 2

# Software installation and test

The installation procedure depends on the plateform you are using. Please refer to the NOMAD website for downloading *Windows*, *Linux*/ *Unix*, or *Mac OS X* versions and to the OPTI Toolbox website for downloading the *Matlab* version (*Windows* only).

This chapter contains the installation procedures for the supported plateforms and describes the content of the installation package. Please refer to the section that fits your needs.

## 2.1 *Windows* installation

For *Windows*, installation requires administrative rights. To start the installation, double-click on the downloaded file (`NOMAD_setup.exe`), and follow the instructions.

**Warning.** The NOMAD executable have been generated with *Microsoft Visual C++ 2010*. Hence, the installation procedure may request the installation of *Visual Studio 2010 runtime redistributables*. If *Microsoft Visual Studio 2010* (Professional or Express edition) is already installed on your computer, this operation may be skipped. Please note that installing *Visual Studio 2010 runtime redistributables* along with *Visual Studio 2010* pauses no problem.

*Windows* environment variables: `%NOMAD_HOME%` and `%NOMAD_EXAMPLES%`

**Defining environment variables** allows a more convenient access to NOMAD. *Windows* environment variables are set automatically during installation. The `%NOMAD_HOME%` environment

variable contains the path to the installation directory. Also, depending on the options selected during installation, a directory containing examples and a copy of the source codes may have been created along with a `%NOMAD_EXAMPLES%` environment variable that contains the path to this examples directory.
However, the examples directory and the corresponding environment variable are set only for the current user account during installation. Hence, other users of the computer need to perform two additional operations after installation is completed:

- Copy the `%NOMAD_HOME%\examples` directory to a convenient location. This is because the `%NOMAD_HOME%` is a read-only directory and running examples requires write permission.
- Set the `%NOMAD_EXAMPLES%` environment variable to where the examples have been copied. This is achieved through the `|Control Panel|System|Advanced|Environment variables|` start menu or by searching for 'environment variable'. First, click on the button to create a new environment variable for the current user. Then add the name `NOMAD_EXAMPLE` and put in the 'variable' field the path where examples have been copied.

*Windows* installation for additional users requires some extra operations

**Testing the installation.** Depending on the option selected during the installation, this is alternatively done by

1. Double clicking on the *NOMAD.3.6.1* icon on the desktop.
2. Or click on the *NOMAD.3.6.1* icon in the *NOMAD.3.6.1* Start Menu.
3. Or start a `cmd` shell window and type
   `"%NOMAD_HOME%\bin\nomad.exe" -info`

This test displays NOMAD general information as in Figure 2.1.

NOMAD binaries are copied during the installation. Nevertheless, information on how to recompile the source codes is provided in Section 4.3.1.

## 2.2 *Mac OS X* installation

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 with no blank space in the name to ease the creation of environment variables. Also choose directories for which you have the adequate write permissions.

**Defining environment variables** allows more convenient access to NOMAD. The first variable to be defined should be `$NOMAD_HOME`, whose value is the path to the directory where

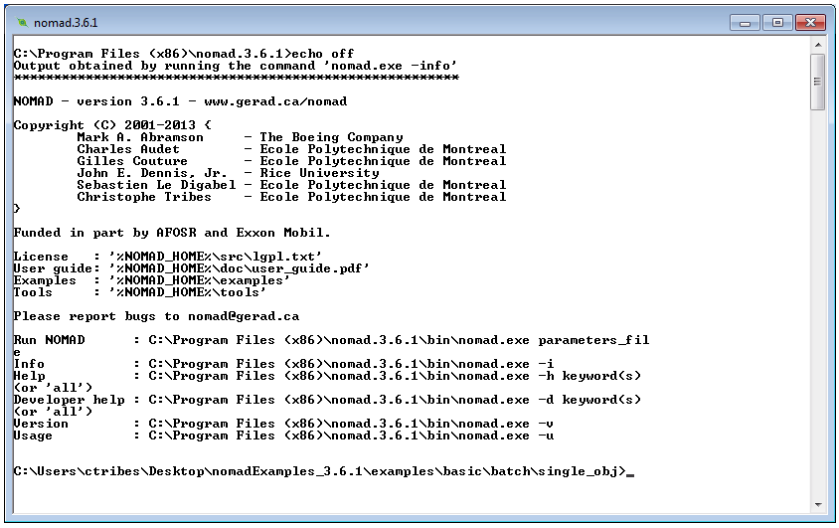


Figure 2.1: Result of the installation test for *Windows*.

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` at the command prompt to execute NOMAD.
Here are some examples on how to modify your environment variables according to the shell you are using:

**For bash shell**, add the following lines in the file `.profile` located in your home directory:

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

---

**For csh or tcsh shell**, add the following lines to the file `.login`:

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

---

To activate the variables, at the command prompt, enter the command `source .profile` or `source .login`, or simply log out and log in.

*Unix/Linux/Mac OS X* `$NOMAD_HOME` is **required for compiling examples**

**Compilation** of source codes is possible (but not required for basic usage with *Mac OS X*10.5 or higher and 64-bit intel processor) if the *gcc* compiler is installed on the machine (if not, install

*Xcode* from Apple developper Web site):

> In a terminal window, do `cd $NOMAD_HOME/install` and execute the `./install.sh` command.
>
> ---
>
> This script automatically compiles the code and generates the NOMAD executable in `$NOMAD_HOME/bin` and the NOMAD library in `$NOMAD_HOME/lib`.
>
> ---
>
> The script also detects if *MPI* is installed. If so, the parallel NOMAD executable and library are generated in the same directories as the scalar version.

**Test the installation** in a terminal window by entering:

> `nomad -info` at the command prompt. The output of the command should be similar than the one depicted by Figure 2.2

```
> nomad -info

NOMAD - version 3.6.1 - www.gerad.ca/nomad

Copyright (C) 2001-2013 {
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

Run NOMAD      : nomad parameters_file
Info           : nomad -i
Help           : nomad -h keyword(s) (or 'all')
Developer help : nomad -d keyword(s) (or 'all')
Version        : nomad -v
Usage          : nomad -u
```

Figure 2.2: Output obtained when testing the installation on *Linux* / *Unix* and *Mac OS X*.

## 2.3 *Unix* and *Linux* installation

For *Unix* and *Linux*, decompress the downloaded zip file where you want to install NOMAD.

**Defining environment variables** allows more convenient access to NOMAD. The first variable to be defined should be `$NOMAD_HOME`, whose value is the path to the directory where NOMAD has been installed. This variable is used by the makefiles provided in the examples and is assumed to be defined in this document. Another environment variable to set is the `$PATH` variable where `$NOMAD_HOME/bin` should be added. This way, you may just type `nomad` at the command prompt to execute NOMAD. Here are some examples on how to modify your environment variables according to the shell you are using:

*Unix/Linux/Mac OS X* `$NOMAD_HOME` is **required for compiling examples**

**For bash shell**, add the following lines in the file `.profile` located in your home directory:

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

---

**For csh or tcsh shell**, add the following lines to the file `.login`:

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

---

To activate the variables, at the command prompt, enter the command `source .profile` or `source .login`, or simply log out and log in.

**Compilation** of source codes must be performed to obtain NOMAD binaries (executables and libraries).

In a terminal window, do `cd $NOMAD_HOME/install` and execute the `./install.sh` command.

---

This script automatically compiles the code and generates the NOMAD executable in `$NOMAD_HOME/bin` and the library in `$NOMAD_HOME/lib`.

---

The script also detects if *MPI* is installed. If so, the parallel NOMAD executable and library are generated in the same directories as the scalar version.

**Test the installation** by entering:

> `nomad -info` at the command prompt. The output of the command should be similar than the content of Figure 2.2

## 2.4 *Matlab* for *Windows* installation

A *Matlab* version of NOMAD can also be downloaded at the OPTI Toolbox website. Please note that the package contains binaries only for *Windows*.[1]

The overall installation including all OPTI provided solvers can be tested by running the test at the command prompt. After installation is completed successfully, NOMAD can be tested alone by typing `nomad('-v')` or `nomad('-info')`. Please note that the version available from the OPTI Toolbox website and from the NOMAD website may differ.

The `nomad` function usage is obtained by typing the command `help nomad`. Please note that parameters are set using a given *Matlab* function called `nomadset`. Usage of this function is described by typing the command `help nomadset`. Once parameters are set they can be passed as an argument of the `nomad` function. Help on parameters can be obtained by typing the command `nomad('-h PARAM_NAME')`. Parameter names are the same as the standalone NOMAD version.

Two NOMAD functionalities are not available in the present *Matlab* version: use of categorical variables and parallel execution.

For additional information concerning installation or utilization please refer to the documentation provided in the OPTI Toolbox package or contact us.

[1] It is possible to compile NOMAD binaries for *Matlab* on *Linux* or *Mac OS X* from NOMAD source codes and the `nomadmex.cpp` main file. Thie procedure may fail due to compatibility issue from *Matlab-MEX* supported and compatible compilers, and the `C++` compiler installed on your machine. Please check this issue before requesting support. For *Matlab 2012a* and *Linux*, information is provided at http://www.mathworks.com/support/compilers/R2012a/glnxa64.html.

## 2.5 Installation directory

Figure 2.3 illustrates the content of the `$NOMAD_HOME` directory once the installation is completed.

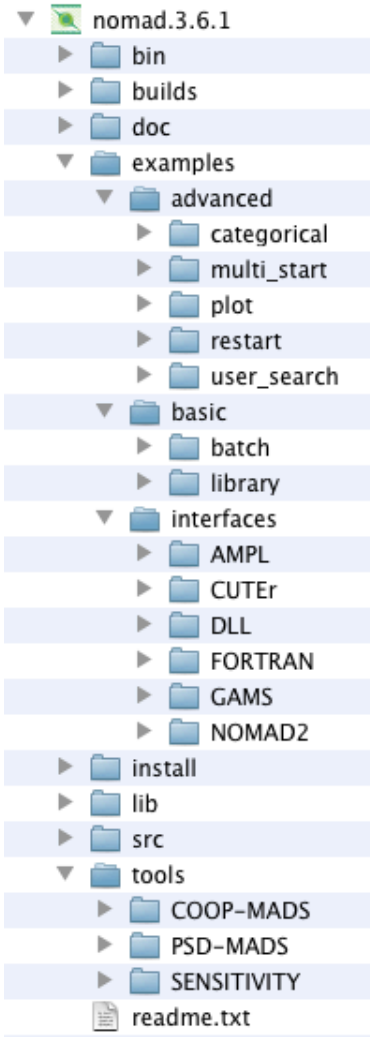


Figure 2.3: Directory structure of the NOMAD package.

# Part II

# BASIC NOMAD USAGE

# Chapter 3

# Getting Started

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

$$\min_{x \in \Omega} f(x) \tag{3.1}$$

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 $\mathbb{R}^n$. The functions $f$ and $c_j$, $j \in J$, 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 in what follows (more details will be provided in Section 4.1), while the library mode allows more flexibility and will be presented in Section 4.3.

This chapter explains how to get started with NOMAD in batch mode. The following topics will be covered:

- How to create a blackbox program.
- How to provide parameters for defining the problem and displaying optimization results.
- How to conduct optimization.

Running the examples provided during the installation requires to have a C++ compiler installed on your machine.

Basic compilation instructions will be provided for *GCC* (the GNU Compiler Collection) and for *Microsoft Visual Studio 2010* (Professional or Express edition).

When using the *Windows* version, it is assumed that NOMAD examples are in a directory for which the user has write permission and the path is `%NOMAD_EXAMPLES%`. This is obtained during installation with default options or can be set afterwards as described in Section 2.1.

## 3.1 How to create blackbox programs

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

In what follows we use the example in the `$NOMAD_HOME/examples/basic/batch/single_obj` (or `%NOMAD_EXAMPLES%\examples\basic\batch\single_obj` for *Windows*). This example optimization problem has a single objective, 5 variables, 2 nonlinear constraints and 8 bound constraints:

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

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

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` (*Linux/Unix/Mac OS X*) or `bb.exe x.txt` (*Windows*). Here `x.txt` is a text file containing a total of $5$ values consisting of **one value for each variable, separated by space(s), tab or linebreak**.
2. The blackbox program returns the evaluation values by displaying them in the **standard output** (default) or by writing them in an output file (see Section 6.2 about advanced parameters). It also **returns an evaluation status of 0** to indicate that the evaluation went well. Otherwise NOMAD considers that the evaluation has failed.
3. The number of values displayed by the blackbox program corresponds to the number of constraints plus one (or two for biobjective 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 $c_2(x) = x_1 + x_2 - 10$).

Blackbox programs must comply with **NOMAD format**

The blackbox `C++` program of the previous example to evaluate the objective and the two constraints for a given design vector is given in Figure 3.1.

**With *GNU* compiler `gcc`,** the blackbox compilation and test are as follows:

1. Change directory to `$NOMAD_HOME/examples/basic/batch/single_obj`.
2. Compile the blackbox program with the following command `g++ -o bb.exe bb.cpp`.
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 ($c_2(x) = 20 > 0$).

**With *Microsoft Visual C++ 2010*,** the black box compilation and test are as follows:

1. Start the *Microsoft Visual C++ 2010* command prompt window (see in `|Start Menu|Microsoft Visual Studio 2010|`).

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

2. Change directory to `"%NOMAD_EXAMPLES%\examples\basic\batch\single_obj"`.

3. Compile the blackbox program with the following command `cl.exe bb.cpp /EHsc`.

4. Test the executable with the text file `x.txt` containing `'0 0 0 0 0'`, by entering the command `bb.exe x.txt`.

5. 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 verified ($c_2(x) = 20 > 0$).

```
> 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
```

Figure 3.2: Example of a blackbox compilation and execution using GNU Compiler for `$NOMAD_HOME/examples/basic/batch/single_obj`.

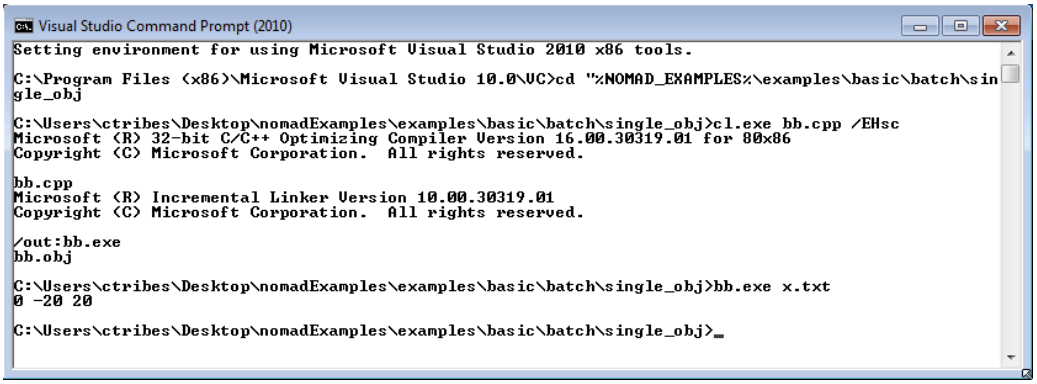


Figure 3.3: Example of a blackbox compilation and execution using *Microsoft Visual C++* Compiler for `%NOMAD_EXAMPLES%\examples\basic\batch\single_obj`.

The order of the displayed outputs must correspond to the order defined in the parameter file (see Section 3.2). If variables have bound constraints, they must be defined in the parameters file and should not appear in the blackbox code.

## 3.2 How to provide parameters

Batch mode parameters are provided in a file as `KEYWORD argument(s)`

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

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

Help on parameters is accessible at the command prompt:

`$NOMAD_HOME/bin/nomad -h param_name` (*Linux*/*Mac OS X*/*Unix*).

`"%NOMAD_HOME%\bin\nomad.exe" -h param_name` (*Windows*).

The two constraints defined in the parameters file in Figure 3.4 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. 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.

```
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
```

Figure 3.4: 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 syntaxic rules to provide parameters.

## 3.3 How to conduct optimization

Optimization is conducted by starting NOMAD executable in a command window with the parameter file name given as argument. To illustrate the execution, the example provided in `$NOMAD_HOME/examples/basic/batch/single_obj/` is considered:

`$NOMAD_HOME/bin/nomad param.txt` (*Linux/Mac OS X/Unix*)

`"%NOMAD_HOME%\bin\nomad.exe" param.txt` (*Windows*)

The outputs are provided in Figures 3.6 and 3.5.

Depending on the plateform and the compiler used, the **final results for the two runs may**

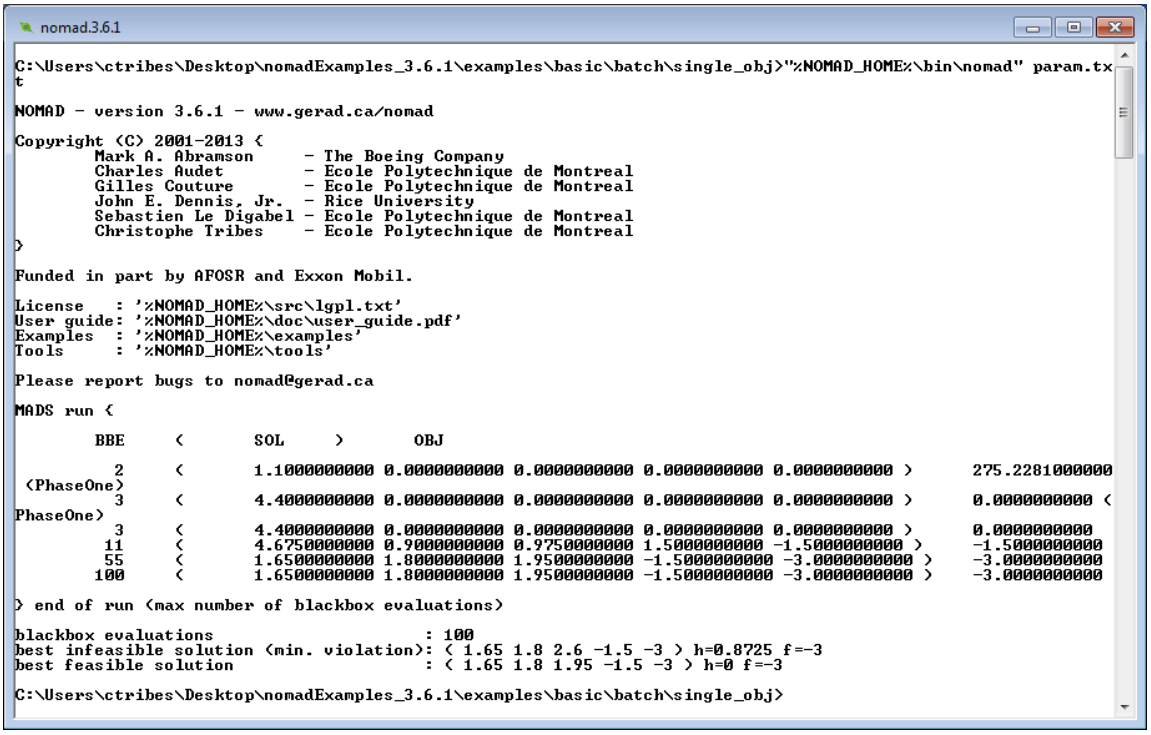


Figure 3.5: Output of NOMAD execution on problem %NOMAD_EXAMPLES%\examples\basic\batch\single_obj (*Windows*).

**differ**. This discrepancy is due to some of the default settings of NOMAD that introduce a non-deterministic behavior but is in average good for optimization performance.

```
> cd $NOMAD_HOME/examples/basic/batch/single_obj
> ls
bb.cpp  bb.exe  param.txt  x.txt
> $NOMAD_HOME/bin/nomad param.txt

NOMAD - version 3.6.1 - www.gerad.ca/nomad

Copyright (C) 2001-2013 {
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 ( SOL ) OBJ

  2 ( 1.1000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000 ) 275.2281000000 (PhaseOne)
  3 ( 4.4000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000 ) 0.0000000000 (PhaseOne)
  3 ( 4.4000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000 ) 0.0000000000
 11 ( 4.6750000000 0.9000000000 0.9750000000 1.5000000000 -1.5000000000 ) -1.5000000000
 55 ( 1.6500000000 1.8000000000 1.9500000000 -1.5000000000 -3.0000000000 ) -3.0000000000
100 ( 1.6500000000 1.8000000000 1.9500000000 -1.5000000000 -3.0000000000 ) -3.0000000000

} end of run (max number of blackbox evaluations)

blackbox evaluations                     : 100
best infeasible solution (min. violation): ( 1.65 1.8 2.6 -1.5 -3 ) h=0.8725 f=-3
best feasible solution                   : ( 1.65 1.8 1.95 -1.5 -3 ) h=0 f=-3
```

Figure 3.6: Output of NOMAD execution on problem `$NOMAD_HOME/examples/basic/batch/single_obj` (*Linux/Mac OS X/Unix*).

# Chapter 4

# How to use NOMAD

This chapter describes how to use NOMAD for solving blackbox optimization problems. Functionalities of NOMAD that are considered more advanced such as biobjective optimization, categorical variables, sensitivity analysis and parallel modes are presented in Chapter 7.

New users are encouraged to first read the Getting Started chapter to understand the basics of NOMAD utilization.

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 Section 4.3.

NOMAD should be cited with references [5, 50]. Other relevant papers by the developers are accessible through the NOMAD website www.gerad.ca/nomad.

## 4.1 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 Section 4.3).

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

1. 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.
2. 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 the **NOMAD specifications** (see Section 3.1 for blackbox basics and Section 4.2.1 for more details).
3. Create a parameters file, for example `param.txt`. This file can be located in the problem directory or not (see Section 4.2 for more details).
4. In the problem directory, start the optimization with a command like `$NOMAD_HOME/bin/nomad param.txt` (*Linux*/*Mac OS X*/*Unix*) or `"%NOMAD_HOME%\bin\nomad.exe" param.txt` (*Windows*).

## 4.2 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

`$NOMAD_HOME/bin/nomad -h` (*Linux*/*Mac OS X*/*Unix*) or
`"%NOMAD_HOME%\bin\nomad.exe" -h` (*Windows*), to see all parameters, or
`$NOMAD_HOME/bin/nomad -h PARAM_NAME` (*Linux*/*Mac OS X*/*Unix*) or
`"%NOMAD_HOME%\bin\nomad.exe" -h PARAM_NAME` (*Windows*) 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. Characters '-', '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. Additional information about parameters and algorithms are provided in Subsection 4.2.1. The advanced parameters and special functionalities of NOMAD are presented in Chapters 6 and 7.

### Problem parameters

| name | arguments | description | default |
|---|---|---|---|
| BB_EXE | list of strings; see 4.2.1 | blackbox executables (required in batch mode) | none |
| BB_INPUT_TYPE | see 4.2.1 | blackbox input types | * R (all real) |
| BB_OUTPUT_TYPE | see 4.2.1 | blackbox output types (required) | none |
| DIMENSION | integer | $n$ the number of variables (required, $n \leq 1000$) | none |
| LOWER_BOUND | see 4.2.1 | lower bounds | none |
| UPPER_BOUND | see 4.2.1 | upper bounds | none |

### Algorithmic parameters

| name | arguments | description | default |
|---|---|---|---|
| DIRECTION_TYPE | see 4.2.1 | type of directions for the poll | ORTHO |
| F_TARGET | reals, f or (f1 f2) | NOMAD terminates if $f_i(x_k) \leq$ fi for all objective functions | none |
| INITIAL_MESH_SIZE | see 4.2.1 | $\Delta_0^m$ [18] | r0.1 or based on X0 |
| LH_SEARCH | 2 integers: p0 and pi | LH (LATIN-HYPERCUBE) SEARCH (p0: initial, pi: iterative); see 4.2.1 (7.2 for biobjective) | none |
| MAX_BB_EVAL | integer | maximum number of blackbox evaluations; see 7.2 for biobjective | none |
| MAX_TIME | integer | maximum wall-clock time (in seconds) | none |
| TMP_DIR | string | temporary directory for blackbox i/o files; see 4.2.1 | problem directory |
| X0 | see 4.2.1 | starting point(s) | best point from a cache file or from an initial LH SEARCH |

### Output parameters

| name | arguments | 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 $[0;2]$ or a string with four digits; see 4.2.1 | 0: no display; 2: full display | 1 |
| DISPLAY_STATS | list of strings | what informations is displayed at each success; see 4.2.1 | see 4.2.1 |
| HISTORY_FILE | string | file containing all trial points with format ( x1 x2 ... xn ) on each line; includes multiple evaluations | none |
| SOLUTION_FILE | string | file to save the current best feasible point or the final best infeasible point if no feasible point has been obtained followed by a warning message | none |
| STATS_FILE | a string file_name plus a list of strings | the same as DISPLAY_STATS but for a display into file file_name | none |

### 4.2.1 Additional information for some parameters

**Parameters** BB_OUTPUT_TYPE **and** 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.
```

The management of blackbox program path containing spaces can be done using special character '$':

```
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 bb.exe
                                   # in nice mode on X systems
```

#### 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.

For categorical variables (mixed integer variable) please refer to Section 7.1.

#### Blackbox output parameter 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. 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 terminology is used to describe the different types of constraints [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:

| | |
|---|---|
| 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 [17]. |
| NOTHING or - | The output is ignored. |
| OBJ | Objective value to be minimized. |
| PB or CSTR | Constraint treated with Progressive Barrier [20]. |
| PEB | Hybrid constraint PB/EB [22]. |
| STAT_AVG | Average of this value will be computed for all blackbox calls (must be unique). |
| STAT_SUM | 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.

### Bound parameters LOWER_BOUND and UPPER_BOUND

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 $\begin{cases} -5 \leq x_1 \leq 8 \\ -5 \leq x_2 \leq 8 \\ -5 \leq x_3 \leq 8 \\ 0 \leq x_4 \leq 8 \\ x_5 \leq 8 \\ -4 \leq x_6 \leq 8 \\ -4 \leq x_7 \, . \end{cases}$

### Direction type parameter DIRECTION_TYPE

The types of direction correspond to the arguments of parameters DIRECTION_TYPE. Up to 4 strings may be employed to describe one direction type.

These 4 strings are s1 in {ORTHO,LT,GPS}, s2 in {∅,1,2,N+1,2N}, s3 in {∅,STATIC,RANDOM, QUAD,NEG}, and s4 in {∅,UNIFORM}. If only 1,2 or 3 strings are given, defaults are considered for the others. Combination of these strings may describe the following 16 direction types:

| | s1 | s2 | s3 | s4 | direction types |
|---|---|---|---|---|---|
| 1 | ORTHO | 1 | | | OrthoMADS, 1. |
| 2 | ORTHO | 2 | | | OrthoMADS, 2. |
| 3 | ORTHO | | | | OrthoMADS, n+1, quad model for (n+1)th dir. |
| 3 | ORTHO | N+1 | | | OrthoMADS, n+1, quad model for (n+1)th dir. |
| 3 | ORTHO | N+1 | QUAD | | OrthoMADS, n+1, quad model for (n+1)th dir. |
| 4 | ORTHO | N+1 | NEG | | OrthoMADS, n+1, (n+1)th dir=-sum n first dirs. |
| 5 | ORTHO | 2N | | | OrthoMADS, 2n. |
| 6 | LT | 1 | | | LT-MADS, 1. |
| 7 | LT | 2 | | | LT-MADS, 2. |
| 8 | LT | N+1 | | | LT-MADS, n+1. |
| 9 | LT | | | | LT-MADS, 2n. |
| 9 | LT | 2N | | | LT-MADS, 2n. |
| 10 | GPS | BIN | | | GPS for binary variables. |
| 11 | GPS | N+1 | | | GPS, n+1, static. |
| 11 | GPS | N+1 | STATIC | | GPS, n+1, static. |
| 12 | GPS | N+1 | STATIC | UNIFORM | GPS, n+1, static, uniform angles. |
| 13 | GPS | N+1 | RAND | | GPS, n+1, random. |
| 14 | GPS | N+1 | RAND | UNIFORM | GPS, n+1, random, uniform angles. |
| 15 | GPS | | | | GPS, 2n, static. |
| 15 | GPS | 2N | | | GPS, 2n, static. |
| 15 | GPS | 2N | STATIC | | GPS, 2n, static. |
| 16 | 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 16 different direction types may be chosen together by specifying DIRECTION_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.

**Output parameters** DISPLAY_DEGREE

Four different levels of display can be set via the parameter DISPLAY_DEGREE, and these levels may be set differently for four different sections of the algorithm (general displays, search and poll displays and displays for each iteration data). The four different levels can be entered with an integer in $[0; 3]$, but also with the strings NO_DISPLAY, MINIMAL_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.

### 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 $\ell$ [8]. |
| MESH_SIZE | Mesh size parameter $\Delta_k^m$ [18]. |
| OBJ | Objective function value. |
| POLL_SIZE | Poll size parameter $\Delta_k^p$ [18]. |
| SGTE | Number of surrogate evaluations. |
| SIM_BBE | Simulated blackbox evaluations (includes initial cache hits). |
| SOL | Solution, with format iSOLj where i and j are two (optional) strings: i will be displayed before each coordinate, and j after each coordinate (except the last). |
| STAT_AVG | The AVG statistic (argument STAT_AVG of BB_OUTPUT_TYPE). |
| STAT_SUM | The SUM statistic defined by argument STAT_SUM for parameter BB_OUTPUT_TYPE. |
| TIME | Wall-clock time. |
| VARi | Value of variable i. 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:

| | |
|---|---|
| %e | Scientific notation (mantissa/exponent) using e character. |
| %E | Scientific notation (mantissa/exponent) using E character. |
| %f | Decimal floating point. |
| %g | Use the shorter of %e or %f. |
| %G | Use the shorter of %E or %f. |
| %d or i | 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 |
|---|---|---|
| %f | auto | auto |
| %5.4f | 5 | 4 |
| %5f | 5 | auto |
| %.4f | auto | 4 |
| %.f | 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.

The same example with
`DISPLAY_STATS $BBE$ & ( $%5.1fSOL, ) & $%.2EOBJ$`
gives
`$1$ & ( $ 10.3$ , $ 5.6$ ) & $-7.03E+02$`.

In case the user wants to explicitely 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 `STATS_FILE`:
`STATS_FILE output.txt BBE ( SOL ) OBJ`.

### Mesh and poll size parameters

The initial mesh size parameter $\Delta_0^m$ [18] is decided by `INITIAL_MESH_SIZE`. In order to achieve the scaling between variables, NOMAD considers the mesh size parameter for each variable. `INITIAL_MESH_SIZE` may be entered with the following formats:

- `INITIAL_MESH_SIZE d0` (same 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 mesh size provided for variable `i` only)

- INITIAL_MESH_SIZE i-j d0 (initial mesh size provided for variables i to j)

The same logic and format apply for providing the MIN_MESH_SIZE and MIN_POLL_SIZE.

Note that a more explicit scaling method is available with the advanced parameter SCALING (see Section 6.2).

### Latin Hypercube search LH_SEARCH

When using LATIN HYPERCUBE (LH) search (LH_SEARCH p_0 p_1 with p_0 or p_1 different than zero) for 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. For biobjective optimization this has a slightly different meaning (see Section 7.2)

### 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.

### 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 ... v$n$-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
```

## 4.3 Optimization in library mode

The library mode allows to tailor the evaluation of the objectives and constraints within a specialized executable that contains NOMAD static library. For example, it is possible to link your own codes 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 will end it. 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 codes to obtain NOMAD binaries (executable and static library) and how to use static library. Then, details on how to interface your own codes are presented.

### 4.3.1 Compilation of the source code

NOMAD source codes provided during installation are located in `$NOMAD_HOME/src` (*Unix/Linux/Mac OS X*) or in `%NOMAD_EXAMPLES%\VisualStudio2010\src` (*Windows*). 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.

**For *Unix*, *Linux* and *Mac OS X*,** we suggest a compilation procedure using the makefiles provided along with the sources. The makefiles are for *GNU* `gcc` compiler and may need

some modifications depending on your system (C++ compiler and make version). Enter the command make all from a terminal opened in directory $NOMAD_HOME/src. This will create the executable file nomad located in $NOMAD_HOME/bin and the static library file nomad.a in $NOMAD_HOME/lib. If the make command fails, try gmake instead of make.

**For *Windows*,** a console application project for *Microsoft Visual C++ (2010)* (professional or express edition are supposed to be available) is provided for convenience. First, double-click on the %NOMAD_EXAMPLES%\VisualStudio2010\nomad.sln (Microsoft Visual Studio Solution). In the menu Debug (express edition) on in the menu Build (professional) click on Build Solution. This will create nomad.exe and nomad.lib in the %NOMAD_EXAMPLES%\VisualStudio2010\bin and ...\lib directories.

*Windows* users can also perform compilation using the *MinGW* environment. In this case, the same makefiles as for *Unix*, *Linux* and *Mac OS X* can be used within a *MSYS* shell window.

### 4.3.2 Using NOMAD static library

Using the NOMAD routines that are in the pre-compiled NOMAD static library with a C++ program requires building an executable. This is illustrated on the example located in the directory
$NOMAD_HOME/examples/basic/library/single_obj
or in
%NOMAD_EXAMPLES%\examples\basic\library\single_obj.
It is identical to the example shown in Chapter 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 or in %NOMAD_EXAMPLES%\examples.

#### Building with *Microsoft Visual C++ (2010)*

The example is provided as the basic_lib_single_obj project in the nomad solution. First, you must build the nomad solution as described before to create NOMAD static library. Then, to build the example, right-click on the basic_lib_single_obj project in the solution explorer and select build. The resulting executable is located in
%NOMAD_EXAMPLES%\examples\basic\library\single_obj. Execution can be started within *Visual Studio* (professional edition only) or in command shell and will produce the result given in Figure 4.1.

New users of *Microsoft Visual C++ (2010)* are encouraged to get familiar with the software application first and than create their own project based on the example provided. Please

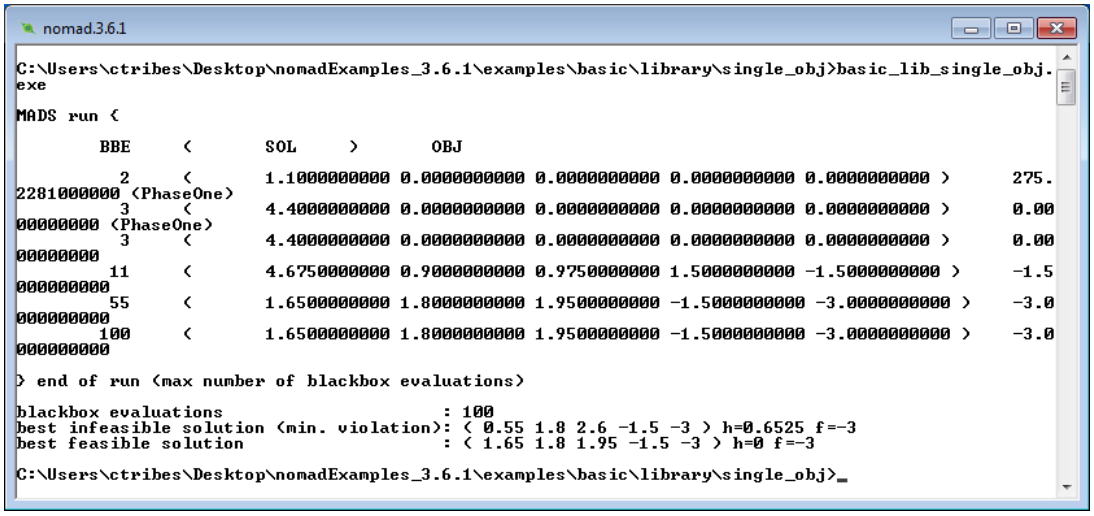


Figure 4.1: Outputs obtained for %NOMAD_EXAMPLES%\examples\basic\library\single_obj (*Windows*).

note that important properties can be modified by right-clicking on a project and selecting Properties.

### Building with makefile

It is supposed that the environment variable $NOMAD_HOME is defined and NOMAD static library is in $NOMAD_HOME/lib. If not, the installation directory of NOMAD must be modified in the makefile. Explanations are given for *GNU* C++ compiler gcc. A basic knowledge of object oriented programmation with C++ is assumed.

Let us first try to compile the basic example. In a terminal, change directory to
$NOMAD_HOME/examples/basic/library/single_obj
and type make. The outputs for this examples are given in Figure 4.2

As a first task to create your own executable, a makefile needs to be created for your source code(s). The makefile for the basic example is shown on Figure 4.3. Notice that each line after the symbol ':' has to begin with a tabulation. Such makefiles are given at various places inside the examples directory.

```
> cd $NOMAD_HOME/examples/basic/library/single_obj
> make
   building the scalar version ...
   exe file : basic_lib.exe
> ls
basic_lib.cpp  basic_lib.exe  basic_lib.o  makefile
> ./basics_lib.exe
-bash: ./basics_lib.exe: Aucun fichier ou dossier de ce type
> ./basic_lib.exe

MADS run {

BBE ( SOL ) OBJ

  2 ( 1.1000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000 ) 275.2281000000
  3 ( 4.4000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000 ) 0.0000000000
 11 ( 4.6750000000 0.9000000000 0.9750000000 1.5000000000 -1.5000000000 ) -1.5000000000
 55 ( 1.6500000000 1.8000000000 1.9500000000 -1.5000000000 -3.0000000000 ) -3.0000000000
100 ( 1.6500000000 1.8000000000 1.9500000000 -1.5000000000 -3.0000000000 ) -3.0000000000

} end of run (max number of blackbox evaluations)

blackbox evaluations    : 100
best infeasible solution: ( 0.55 1.8 2.6 -1.5 -3 ) h=0.6525 f=-3
best feasible solution  : ( 1.65 1.8 1.95 -1.5 -3 ) h=0 f=-3
```

Figure 4.2: Outputs obtained for `$NOMAD_HOME/examples/basic/library/single_obj` (*Linux*).

### 4.3.3 Definition of the blackbox evaluation

We now describe the other steps required for the creation of the source file (let us call it `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` for *Linux/Unix/Mac OS X*) or with build command in *Visual Studio C++ (2010)*, the binary file (let us call it `basic_lib.exe`) is created and can be executed.

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 class `NOMAD::Point`. A lot of functionalities have been coded for theses 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

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

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` 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 4.5.

The blackbox evaluation is programmed in a user-defined class. The objective of this user class is to redefine the virtual method `NOMAD::Evaluator::eval_x()` that will be automatically called by the algorithm.

The prototype of `eval_x()` is given in Figure 4.4. Note that `const` and non-`const` versions of the method are available. The function `eval_x()` should return `true` if the evaluation

succeeded, and `false` if the evaluation failed.

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

Figure 4.4: Prototype of method `NOMAD::Evaluator::eval_x()`. A non-const version is also 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 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. Please refer to Section 6.1 for description of parameters $h_{max}$ and $h_{min}$.

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`).

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 explicitely set the flag `count_eval` to `true`. Additionnaly, 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 whithin `eval_x()`.

The virtual method `NOMAD::Evaluator::update_success()` can also be subclassed in the

```
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 4.5: Example of a user class defining a hard-coded blackbox evaluation.

`My_Evaluator` class. 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()`.

### 4.3.4 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.

### Setting parameters

In library mode, parameter PNAME is set with set_PNAME().

In library mode, the main function must declare a NOMAD::Parameters object. Parameter names are the same as in batch mode but may be defined programmatically. A parameter PNAME is set with the method NOMAD::Parameters::set_PNAME().

In order to see all the options attached to a parameter PNAME, use the help
$NOMAD_HOME/bin/nomad -h PNAME (*Linux/Unix/Mac OS X*) or
"%NOMAD_HOME%\bin\nomad.exe" -h PNAME (*Windows*) or
refer to the list of parameters in Sections 4.2 and 6.1, or to the header file Parameters.hpp.

In library mode it is also possible to provide the parameters 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;.

NOMAD types can be used as arguments of NOMAD::Parameters functions. An example is given in Figure 4.6. 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 3.4 except for BB_EXE which is not required.

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.

If parameters are changed after a first check, 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.

#### 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.

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 4.7.

In the example described in Figure 4.7, 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`.

#### Access to solution and 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;
```

### 4.3.5 Other functionalities of the library mode

#### 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). See $NOMAD_HOME/src/Evaluator.hpp for the header of this method.
- 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.

### 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 4.8. 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).

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 4.9.

Examples showing multiple MADS runs are provided in the $NOMAD_HOME/examples/advanced/restart and $NOMAD_HOME/examples/advanced/multi_start directories. The multistart program launches multiple instances of MADS with different starting points from a LATIN-HYPERCUBE sampling.

## 4.4 Interface examples

Blackbox evaluations programs can be in different format. The example directory `$NOMAD_HOME/examples/interfaces` illustrates how to interface NOMAD in the following cases:

- **AMPL**: The 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* [41] 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* [33] program.
- **NOMAD2**: Program to use NOMAD version 3 on a problem originally designed for the version 2. This example has been written by Quentin Reynaud.

```
// 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 4.6: 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 4.7: `Evaluator` and `Mads` objects usage.

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

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

// some changes...

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

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

Figure 4.8: Two runs of MADS with a `NOMAD::Mads` object at local scope. The cache is erased between the two runs.

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

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

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

Figure 4.9: Two runs of MADS with a `NOMAD::Mads` object at a more global scope. The cache is kept between the two runs.

# Chapter 5

# Tricks of the trade

NOMAD has default values for all 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.

## 5.1 Tune NOMAD

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

| Symptom | Suggestion | Ref |
|---|---|---|
| I want to see more display | Set display degree | page 39 |
| Blackbox manage output file name | Use `BB_REDIRECTION` | see 6.2 |
| Quantifiable constraints | Try PB, EB, PEB or combinations | page 36 |
| Difficult constraint | Try PB instead of EB | 4.2.1 |
| | Relax feasibility criterion `H_MIN` | page 64 |
| No initial point | Add a LH search | 4.2.1 |
| Variables of widely different magnitudes | Provide scaling | 6.2 |
| | Change $\Delta_0$ per variable | page 41 |
| | Tighten bounds | page 38 |
| Many variables | Fix some variables | 6.2 |
| | Use PSD-MADS | 7.3 |
| Unsatisfactory solution | Change initial point | page 42 |
| | Add a VNS search | 7.5 |
| | Add a LH search | page 42 |
| | Change direction types | page 38 |
| | Deactivate `SNAP_TO_BOUNDS` | page 65 |
| | Tighten bounds | page 38 |
| | Change $\Delta_0$ | page 41 |
| | Modify seeds that affect algorithms | page 67 |
| | Disable models | page 64 |
| Improvements get negligible | Change stopping criteria | Type `nomad -h stop` |
| | Disable models | page 64 |
| It takes long to improve $f$ | Decrease $\Delta_0$ | page 41 |
| Optimization is time consuming | Perform parallel blackbox evaluations | 7.3 |
| | Use surrogate | 6.2 |
| | Provide a user search | 7.6 |
| Blackbox is not that expensive | Setup maximum wall-clock time | 4.2 |
| | Add a VNS search | 7.5 |
| | Add a LH search | 4.2.1 |
| | Combine direction types | page 38 |
| Biobjective optimization slow | Treat an objective as a constraint | 4.2.1 |
| | Perform parallel blackbox evaluations | 7.3 |
| | Set bounds for objectives | 4.2.1 |
| Biobjective optimization stops prematurely | Increase evaluation budget per MADS run | 7.2 |
| | Decrease number of MADS run | 7.2 |
| | Relax feasibility criterion `H_MIN` | 6.1 |

## 5.2 Dynamically plot optimization history

Users may want to plot information during NOMAD execution. This can be achieved by implementing the `NOMAD::Evaluator::update_success()` virtual function. You can find an example using *Java* `$NOMAD_HOME/examples/advances/plot`.

## 5.3 Tools to visualize results

What-if scenarios, and sensitivity to constraints can be post analyzed with the tools of Section 7.4.

## 5.4 Use categorical variables

My variables can be represented by integers, but the numbers do not mean anything and they cannot be logically ordered without further analyses. Perhaps your problems would be more suitably represented using categorical variables. In particular, when your problem has a number of design variables that can vary by selecting a parameter, this parameter can be set as a categorical variable. See Section 7.1.

# Part III

# ADVANCED NOMAD USAGE

# Chapter 6

# Advanced parameters

## 6.1 Parameters description

Advanced parameters are intended to setup optimization problems, algorithmic and output parameters when specific needs are present.

### Problem parameters

| name | arguments | description | default |
|---|---|---|---|
| FIXED_VARIABLE | see 6.2 | 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 | $\infty$ |
| SGTE_EVAL_SORT | bool | if surrogates are used to sort list of trial points | yes |
| SGTE_EXE | list of strings; see 6.2 | surrogate executables | none |
| VARIABLE_GROUP | index range | defines a group of variables; see 6.2 | none |

## Algorithmic parameters

| name | arguments | description | default |
|---|---|---|---|
| ASYNCHRONOUS | bool | asynchronous strategy for the parallel version; see 7.3 | 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 6.2 | yes |
| CACHE_SEARCH | bool | enable or disable the cache search (useful with extern caches) | no |
| DISABLE | string | forcefully disables a feature provided as argument; see 6.2 | - |
| 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 7.1 | r0.1 |
| HALTON_SEED | integer | Halton seed for ORTHO-MADS [8] | $n$th prime number |
| 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 $h(x) \leq$ 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 $\ell_0$ [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 $\ell_{max}$ [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 | $\tau$ [18] | 4.0 |
| MIN_MESH_SIZE | see 4.2.1 | $\Delta^m_{min}$ [18] | none |
| MIN_POLL_SIZE | see 4.2.1 | $\Delta^p_{min}$ [18] | none or 1 for int/bin variables |
| 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 |
| MODEL_SEARCH_OPTIMISTIC | bool | if model search is optimistic or not | yes |
| MULTI_F_BOUNDS | 4 reals | see 7.2 | none |
| MULTI_NB_MADS_RUNS | integer | number of MADS runs | see 7.2 |
| MULTI_OVERALL_BB_EVAL | integer | max number of blackbox evaluations for all MADS runs | see 7.2 |
| NEIGHBORS_EXE | string | neighborhood executable for categorical variables in batch mode | none |
| OPPORTUNISTIC_CACHE_SEARCH | bool | opportunistic strategy for cache search | no |
| OPPORTUNISTIC_EVAL | bool | opportunistic strategy; see 6.2 | yes |
| OPPORTUNISTIC_LH | bool | opportunistic strategy for LH search; see 7.2 for biobjective | see 6.2 |
| OPPORTUNISTIC_MIN_EVAL | integer | see 6.2 | none |
| RHO | real | $\rho$ parameter of the Progressive Barrier | 0.1 |
| SCALING | see 6.2 | scaling on the variables | none |
| SEED | integer or NONE | random seed; NONE or a negative integer to define a seed that will be different at each run | NONE |
| 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 generates a feasible solution | no |
| USER_CALLS_ENABLED | bool | if no, the automatic calls to user functions are disabled | yes |
| VNS_SEARCH | bool or real | VNS search; see 7.5 | no |

### Ouptut parameters

| 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 | '-' |

## 6.2 Detailed information for some parameters

Detailed information for some of the parameters are provided in alphabetical order.

#### 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. Alternatively, the output file name can be obtained from the input file name by replacing the extension 'input' by 'output'.
In addition, TMP_DIR can be specified by the user. If no, the default TMP_DIR is the problem directory.

### Parameter DISABLE

The DISABLE parameter is used to forcefully disable a feature. Currently, only MODELS is accepted as argument. DISABLE MODELS is equivalent to set: MODEL_EVAL_SORT no, MODEL_SEARCH no and DIRECTION_TYPE ORTHO N+1 NEG (if direction type is set to ORTHO N+1 QUAD, that is the default). Please note that this parameter as no default ant that extra settings of MODEL_EVAL_SORT, MODEL_SEARCH and DIRECTION_TYPE ORTHO N+1 QUAD will be ignored.

### 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 ... v$n$-1). Again, character '-' may be used for free variables.
- An index range if at least one starting point has been defined (see 4.2.1 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).

### Parameters HALTON_SEED and SEED

The directions that NOMAD explores during the poll phase are dependent upon the Halton seed. The Halton seed is used to generate a pseudo-random sequence of numbers. The user can change the sequence of directions by setting HALTON_SEED to a selected value.

Other aspects of NOMAD may depend on a pseudo-random sequence of numbers depending on selected options: LH Search, GPS and LT directions, evaluation order priority, BiMads and categorical variables. The sequence of numbers is controlled by SEED to a selected value.

### Opportunistic strategies OPPORTUNISTIC_EVAL, OPPORTUNISTIC_CACHE_SEARCH and OPPORTUNISTIC_LH

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 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 7.2).

### 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 (that is scaled) before an evaluation and the call to NOMAD::Evaluator::eval_x(). Outside of this method the variables are unscaled.

All NOMAD outputs (including files) display unscaled values. All variable-related parameters (bounds, starting points, fixed variables) must be specified as unscaled. 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_HOME/bin/nomad -h scaling for more details about the use of SCALING.

### Executable parameters SGTE_EXE

*Surrogates*, or *surrogate functions*, are cheaper blackbox functions that are used, at least partially, to drive the optimization (see Figure 6.1).

The current version of NOMAD uses only static surrogates which are not updated during the algorithm and which are provided by the user. See [32] for a survey on surrogate optimization.

In batch mode, 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,
- SGTE_EXE sgte_exe when only one blackbox executable is used. Surrogates must display the same number of outputs as their associated blackboxes.

In the library mode, if a surrogate function is to be used, then its evaluation routine should be coded in the method eval_x() (see Section 4.3.3). First, to indicate that a surrogate can be

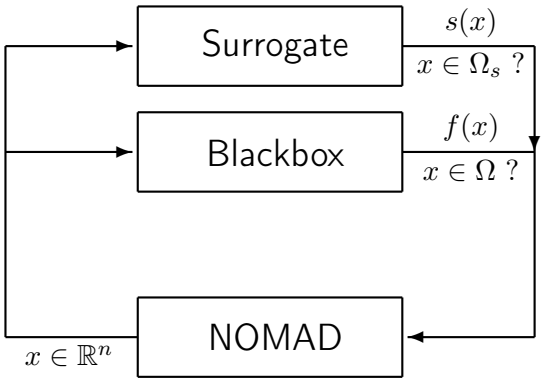


Figure 6.1: Blackbox optimization using surrogates.

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.

#### Group of variables parameter `VARIABLE_GROUP`

By default NOMAD creates one group that combines all continuous, integer, and binary variables, and one group for categorical variables (see Section 7.1).

In batch mode, the `VARIABLE_GROUP` parameter followed by variable indices is used to explicitely 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 (except for categorical variables). If a group contains only binary variables, directions of type `NOMAD::GPS_BINARY` will be automatically used.

To define some particular types of directions or a particular Halton seed for the groups (for OrthoMADS directions only), use the NOMAD library 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 set of indices of the variables composing each group is provided as argument of the function. 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.

# Chapter 7

# Advanced functionalities

## 7.1 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.

Examples on categorical variables for a simple portfolio selection problem are provided in `$NOMAD_HOME/examples/advanced/categorical`. A single-objective and a biobjective version are given in library and batch mode.

The algorithm used by NOMAD to handle mixed variables problems is defined in references [1, 4, 7, 15, 49] and works as follows.

### 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.

### 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 neighboors 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 prob-

lem. 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 charateristics 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 charateristics (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 4.2.1) because it has no reference signature for defining a value for each categorical variable.

Changing the values of the categorical variables is done exclusively during the extended poll phase by providing the neighbors of the current iterate. The logic for providing the neighboors is entirely left to the user. For this reason, it is not necessary to provide bounds for the categorical variables whether in the initial description of the problem or when providing extended poll point signatures. A warning message may be displayed when providing bounds for categorical variables.

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 $f(y) < f(x_k)+$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::Extended_Poll::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 to their

initial value.

## 7.2 Biobjective optimization

NOMAD can solve biobjective optimization problem in search of a Pareto front. Examples of biobjective problems solved by NOMAD in library and batch mode are given in `$NOMAD_HOME/basic/(batch|library)/bi_obj`.

NOMAD performs biobjective optimization through the BiMADS algorithm described in [27]. The BiMADS algorithm solves biobjective problems of the form

$$\min_{x \in \Omega} \qquad F(x) = \big(f_1(x), f_2(x)\big). \tag{7.1}$$

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 [27], 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 $\delta$ 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_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.

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. In this case, the criterion defines a utopian point. If it can be achieved than the pareto front is a single point.
- The 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 $\delta$ 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 $\delta$), then the number of MADS runs should be larger, with less severe stopping criteria on single-objective optimizations.

## 7.3 Parallel versions

### 7.3.1 Compilation

The NOMAD parallel versions are based on the message passing interface (*MPI* [58]). In particular, the *MPI* implementations *openMPI*, *LAM*, *MPICH* and the *Microsoft HPC pack* have been tested. To obtain the parallel binaries (executable and static library), NOMAD must link with *MPI*.

**For *Linux*, *Unix* or *Mac OS X*,** the parallel binaries can be obtained by typing `make mpi` in the `$NOMAD_HOME/src` directory, after ensuring that the command `mpic++` works. The compilation will produce the executable `nomad.MPI.exe` and the static library `nomad.MPI.a`. It is also possible to install all NOMAD binaries (all combinations of scalar/parallel and executables/static libraries) by typing `./install.sh` in the `$NOMAD_HOME/install` directory.

**For *Windows*,** the parallel binaries can be obtained by compiling NOMAD with *Microsoft Visual C++ (2010)*. 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 the *MPI* library directory into `'Linker Additional Library Directories'`, and add the *MPI* library (typically `mpi.lib`) into `'Linker Input Additional Dependencies'`. Finally, add the location of the *MPI* header file into 'Additional Include Directories'. Please note that it is also possible to obtain the parallel binaries by using the *GNU* `C++` compiler with the *MinGW* environment. Details to perform parallel compilation/configuration/execution are provided in the document `%NOMAD_HOME%\install\readme.MPI_for_MINGW.rtf`.

### 7.3.2 Algorithms

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 linked with the NOMAD static library (scalar), and are located in the `tools` directory. These parallel versions are developed with *MPI* [58] 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 6.2).

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 [51].

For the sake of simplicity, the remaining of the discussion focuses on utilizing the parallel version in the *Linux/Unix/Mac OS X* environments. The same principles apply in the *Windows* environment but the tasks can be performed via the *Microsoft Visual C++* menus and in the command shell window. Please contact NOMAD support if more details are needed.

### 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, chosing `np=7` may be a better choice than `np=9`.

### 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 static library. Makefiles working for *Linux*, *Unix* and *Mac OS X* 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.

### The PSD-MADS method

The PSD-MADS method implements a parallel space decomposition of MADS and is is described in [21]. The method aims at solving larger problems than the scalar version of NOMAD.

NOMAD is in general efficient for problems with up to $\simeq$ 20 variables, PSD-MADS has solved problems with up to 500 variables.

In PSD-MADS, each slave process has the responsibility for a small number of variables on which aMADS 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 [51], 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 CPU ressources).

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

## 7.4 Sensitivity analysis

Sensitivity analysis can perform **'What If'** studies in engineering problems context.

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.

Details on the sensitivity analysis with blackboxes and some theoretical results on a smooth case may be consulted in [23].

Two tools are available in directory $NOMAD_HOME/tools/SENSITIVITY as program sources and can be compiled with makefiles. The tools generate the data necessary to plot objective versus constraint graphs.

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 [27]. 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

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

where $\Omega_j$ is the feasible set $\Omega$ minus the constraint. The use of the BiMADS algorithm allows to focus explicitely 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 $\overline{c}_j$ 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 preview of the sensitivity without additional blackbox evaluation. 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.

## 7.5 Variable Neighborhood Search (VNS)

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

The VNS search strategy is described in [12]. It is based on the Variable Neighborhood Search metaheuristic [55, 45].

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 NOMAD the VNS search strategy is not activated by default. In order to use the VNS search, 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.

## 7.6 User search

The default search strategy in NOMAD is based on quadratic models. But, users may code their own search strategy.

The search must be programmed in a user-defined class. The objective of this user class is to redefine the virtual method NOMAD::Search::search() that will be automatically called by the algorithm. The prototype of search() is given in Figure 7.1.

Users can take the example in $NOMAD_HOME/examples/advanced/user_search to setup their own search. 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.

```
void search
( NOMAD::Mads               & mads           ,
  int                       & nb_search_pts  ,
  bool                      & stop           ,
  NOMAD::stop_type          & stop_reason    ,
  NOMAD::success_type       & success        ,
  bool                      & count_search   ,
  const NOMAD::Eval_Point *& new_feas_inc   ,
  const NOMAD::Eval_Point *& new_infeas_inc   ) ;
```

Figure 7.1: Prototype of method `NOMAD::Search::search()`.

# Part IV

# ADDITIONAL INFORMATION

# Appendix A

# Release notes

## A.1 Version 3.6

**Major changes**

- The OrthoMADS algorithm has been extented to use `N+1` directions in addition of the existing `2N` version. The default setting is `DIRECTION_TYPE ORTHO N+1 QUAD` that makes use of quadratic models to obtain the $(n+1)^{\text{th}}$ direction. Preliminary tests have shown that using the new default setting significantly improves performance over the previous version (OrthoMADS 2N).

- Quadratic models are available for BiMads.

- The *Matlab* version is available.

- The user guide has been reformatted.

- The installation procedure for *Windows* has been modified to allow copy of examples in a user-specified directory.

### Minor changes

A few bugs have been corrected and some minor changes listed below have been applied. The changes from version 3.6.0 to 3.6.1 are identified.

- The final display in version 3.6.1 shows the best infeasible point (with mininum violation) and version 3.6.0 shows the best infeasible point the closer to `h_max`.
- The file provided in `SOLUTION_FILE` contains the current best feasible (version 3.6.0) or the final best infeasible point (with mininum violation) if no feasible point has been obtained (version 3.6.1) followed by a warning message.
- Default direction type is `ORTHO N+1 QUAD` (version 3.6.1) instead of `ORTHO N+1 NEG` (version 3.6.0).
- Fix bug in handling integer and binary variables for quadratic model optimizations and initial mesh size selection (version 3.6.1).
- Fix bug in direction group index that could lead to crash (version 3.6.1).
- Modify the method for rounding integer variables (version 3.6.1).
- Fix bug when using `VNS_SEARCH` with `PEB` type constraints.
- Add a level of display (`NO_DISPLAY, ++MINIMAL_DISPLAY++, NORMAL_DISPLAY, FULL_DISPLAY`).
- Add one category of parameters for developer needs.
- Re-distribute NOMAD parameters among the three categories (basic, advanced, developper).
- Change the implicit variable scaling procedure based on bounds and initial values.
- Add several parameters to control OrthoMADS with `N+1` directions.
- Display a specific message for users attempting to use binary compiled for *MPI* on a single process, that is without using `mpirun`.
- Modify test that $x_0$s are within bounds for biobjective problem and categorical variables.
- Change display rules for phase one search.
- Fix display of evaluation for extented poll with the option `DISPLAY_ALL_EVAL`.

- Add the parameter DISABLE to forcefully disable a feature that is used in several algorithms. Presently, only MODELS can be disabled (used for model search, for sorting before evaluation, for obtaining a $n+1$th direction).
- Fix bug on initial mesh size test.
- Fix bug on a check bimads flag that was reset improperly during phase-one search (bug happen when using VNS_SEARCH.
- Fix bug when using categorical variable with a signature that contains a fixed variable.

## A.2 Previous versions

### Version 3.5

- **Quadratic models** are used to improve the algorithm efficiency. Details and benchmarks are available in [34]. A new model search strategy has been implemented in which a local quadratic model is built and optimized in order to provide up to 4 new trial points at each iteration. Also, with model ordering a local quadratic model is built and the points are sorted accordingly to this model so that the most promising points are evaluated first.
- 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 7.1 and the example located in examples/advanced/categorical/batch.
- A series of parameters influencing the behavior of model search have been renamed for consistency.
- When CTRL-C is pressed an evaluation can be interrupted in library mode within the user provided function eval_x().
- A random number generator have been implemented to allow repeatability of the results on different plateforms.
- A bug in the display format of the stats present when compiling with *Microsoft 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.

### Version 3.4

- **Parallelism**: Three parallel algorithms are now available. See Section 7.3 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 6.2.
- Tool for sensitivity analysis (see Section 7.4).

### Version 3.3

- Handling of **categorical variables** for MIXED VARIABLE PROBLEMS (MVP). See Section 7.1.

### Version 3.2

- VARIABLE NEIGHBORHOOD SEARCH (VNS) described in Section 7.5.
- **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.

### Version 3.1

- **Biobjective optimization**: see Section 7.2.
- **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.

## A.3 Future versions

Future algorithmic developments include:

- Adaptive surrogates and use of the surrogate management framework [32].
- MULTI-MADS: multi-objective variant of MADS [27], with 3 and more objective functions.
- Improving BIMADS algorithm when feasible solutions are difficult to find.
- Better management of integer variables.
- Making surrogate functions available for the *Matlab* version.

# Appendix B

# Developer parameters

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

Please note that the '-d PARAM_NAME' option is required which is different than the '-h PARAM_NAME' option required for other parameters.

| name | arguments | description | default |
|---|---|---|---|
| EPSILON | real | precision on reals | 1E-13 |
| 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_NP1_QUAD_EPSILON | real in $]0;1[$ | truncated unit hypercube $]\epsilon;1[^n$ in quad model optimization | 0.01 |
| 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 interpolation sets for quadratic models | 500 |
| MODEL_QUAD_MIN_Y_SIZE | integer or string | inf. limit on the size of interpolation 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 maintain well-poisedness with quadratic models | no |
| MULTI_FORMULATION | string | how to compute one value from two objectives | PRODUCT or DIST_L2 |
| MULTI_USE_DELTA_CRIT | bool | use stopping criterion based on $\delta$ measure | no |
| OPPORTUNISTIC_LUCKY_EVAL | bool | parameter for opportunistic strategy | none |
| OPPORTUNISTIC_MIN_F_IMPRVMT | real | parameter for opportunistic strategy | none |
| OPPORTUNISTIC_MIN_NB_SUCCESS | integer | parameter for opportunistic strategy | none |
| OPT_ONLY_SGTE | bool | minimize only with surrogates | no |
| SEC_POLL_DIR_TYPE | see 4.2.1 | type of directions for the secondary poll | see 4.2.1 |

# Appendix C

# Statistical dynamic surrogates

In addition to quadratic models, the NOMAD package includes the use of statistical dynamic surrogates constructed with the TGP package [42] based on the treed Gaussian processes developed by Gramacy and Lee [44]. Dynamic surrogates are different from the static surrogates available with the `SGTE_EXE` option: a static surrogate is provided by the user and is never updated, while a dynamic surrogate is automatically constructed and updated by NOMAD. Thus we refer to them as models in order to avoid confusion.

The statistical models obtained with TGP are used in two different places in NOMAD: as a search step, called the model search (option `MODEL_SEARCH`), and for ordering the different lists of trial points before they are evaluated (option `MODEL_EVAL_SORT`).

The sorting of the points consists to evaluate the model on the candidates and then to consider first the most promising points according to the model. This exploits the opportunistic strategy. The principle of the model search is based on the Surrogate Management Framework of Booker et al. [32] and is similar to the use of quadratic models, detailed in [34]: at each iteration, based on the cache points, one model is constructed for the objective function and for each constraint. These models are then optimized by NOMAD itself, which gives new candidates to evaluate. There are differences with quadratic models: first there is no interpolation radius: all points in the cache are considered to construct the model, because TGP aims at global interpolation. Hence, NOMAD and TGP can be viewed as a good tool for global optimization.

Another difference is that TGP provides additional statistics such as the Expected Improvement (EI) [47], which allows a richer exploration of the design space. These different strategies are detailed in the reference [43].

In order to use NOMAD with TGP, the user has to install the external TGP library found here: http://www.cran.r-project.org/web/packages/tgp/index.html. Note that TGP is developed in R [57] which needs to be installed first.

NOMAD uses a dynamic library version of TGP, which needs to be compiled by the command `R CMD INSTALL tgp` executed in top of the TGP installation folder, which should be memorized in the environment variable `$TGP_HOME`. Then NOMAD must be compiled using the provided `makefile.TGP` makefile.

Finally, in order to activate the use of TGP, the parameters file has to include the following instructions:

```
MODEL_SEARCH    TGP
MODEL_EVAL_SORT TGP
```

In this case both the model search and the sorting of the trial points will be performed with TGP. It is however possible to assign one of these two options to either the quadratic models or to nothing.

Finally, TGP is stochastic and its behaviour will be different at each execution, depending on the value of the SEED parameter.

# Bibliography


[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., S. Le Digabel, and C. Tribes. 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. Béchard, and J. Chaouki. Spent potliner treatment process optimization using a MADS algorithm. *Optimization and Engineering*, 9(2):143–160, 2008.

[12] 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.

[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. *Optimization Methods and Software*, 27(4–5):613–624, 2012.

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

[25] C. Audet, S. Le Digabel, and C. Tribes. NOMAD user guide. Technical Report G-2009-37, Les cahiers du GERAD, 2009.

[26] C. Audet and D. Orban. Finding optimal algorithmic parameters using derivative-free optimization. *SIAM Journal on Optimization*, 17(3):642–664, 2006.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

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

[34] A.R. Conn and S. Le Digabel. Use of quadratic models with mesh-adaptive direct search for constrained black box optimization. *Optimization Methods and Software*, 28(1):139–158, 2013.

[35] 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.

[36] 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.

[37] J.E. Dennis, Jr. and V. Torczon. Direct search methods on parallel machines. *SIAM Journal on Optimization*, 1(4):448–474, 1991.

[38] 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.

[39] 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 algorithm. *CALPHAD: Computer Coupling of Phase Diagrams and Thermochemistry*, 36:135–143, 2012.

[40] 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.

[41] 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.

[42] R.B. Gramacy. tgp: An R package for Bayesian nonstationary, semiparametric nonlinear regression and design by treed Gaussian process models. *Journal of Statistical Software*, 19(9):1–46, 2007.

[43] R.B. Gramacy and S. Le Digabel. The mesh adaptive direct search algorithm with treed Gaussian process surrogates. Technical Report G-2011-37, Les cahiers du GERAD, 2011.

[44] R.B. Gramacy and H.K.H. Lee. Bayesian treed Gaussian process models with an application to computer modeling. *Journal of the American Statistical Association*, 103(483):1119–1130, 2008.

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

[46] R.E. Hayes, F.H. Bertrand, C. Audet, and S.T. Kolaczkowski. 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.

[47] D.R Jones, M. Schonlau, and W.J. Welch. Efficient global optimization of expensive black box functions. *Journal of Global Optimization*, 13(4):455–492, 1998.

[48] 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.

[49] 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.

[50] S. Le Digabel. Algorithm 909: NOMAD: Nonlinear optimization with the MADS algorithm. *ACM Transactions on Mathematical Software*, 37(4):44:1–44:15, 2011.

[51] 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 http://www.gerad.ca/Sebastien.Le.Digabel/talks/2010_JOPT_25mins.pdf.

[52] 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.

[53] 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.

[54] 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.

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

[56] M.S. Ouali, H. Aoudjit, and C. Audet. Optimisation des stratégies de maintenance. *Journal Européen des Systèmes Automatisés*, 37(5):587–605, 2003.

[57] R Development Core Team. R*: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2004. http://www.R-project.org.

[58] M. Snir, S.W. Otto, S. Huss-Lederman, D.W. Walker, and J. Dongarra. *MPI: The Complete Reference*. The MIT Press, Cambridge, Massachusetts, 1995.

[59] T.A. Sriver, J.W. Chrissis, and M.A. Abramson. Pattern search ranking and selection algorithms for mixed variable stochastic optimization, 2004. Preprint.

[60] 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.

# General index

# Index of NOMAD parameters