User guide to NOMAD with the GUI

1 – Introduction

NOMAD is a C++ implementation of the Mesh Adaptive Direct Search (MADS) class of algorithm. MADS encompasses the Generalized Pattern Search (GPS) class of algorithm. NOMAD and references to MADS and GPS can be found at the web site:

http://www.gerad.ca/NOMAD/

NOMAD is designed to solve non-linear non-smooth noisy optimization problems of the type:

min_x f(x) subject to C(x) <= 0 and l <= x <= u,

where f is a function from R^n to R, C is a vector function from R^n to R^m, and both l and u are vectors in R^n. Note that m may be zero in the absence of constraints and l or u can be respectively minus infinity or infinity.

Typically, the functions f and C are given as computer codes that read a value of x, and when executed, return a function value. Due to numerical noise, or to the nature of the functions, it is possible that f or C fail to evaluate. For a given x, it is possible for example that the code breaks down and returns a bus error. NOMAD treats these evaluations by setting the function value to infinity. NOMAD is designed to be robust and flexible. All algorithmic parameters have default values, or can be tuned by a user knowledgeable of MADS algorithms.

2 – Installation

2.1 NOMAD extraction

The package you downloaded is called ‘nomad_c6.tar’. Extracting the files will create a directory called ‘NOMAD’ where all the program files will be placed. To extract, type this command in a terminal:

```
tar –xf nomad_c6.tar
```

Once the files are extracted, you’ll see these directories:

- NOMAD is the main directory. It contains the program source code (.h and .cpp files).
- NOMAD/CACHE contains files needed to run NOMAD with the Berkeley database.

- NOMAD/DOC contains the documentation files.
- NOMAD/HS23 contains the source code for a simple test problem, Hock and Schittkowski 23.
- NOMAD/LIBRARY is where NOMAD stores some files, including the history file.
- NOMAD/NOCACHE contains files needed to run NOMAD without the Berkeley database.

2.2 QT installation

NOMAD’s GUI was built using QT, a C++ cross-platform GUI toolkit: QT must be installed on your computer for the GUI to work.

- Unix or MAC OS X users: Download and install the QT package, at this address:
  http://www.trolltech.com/download/qt/x11.html
  Follow QT’s installation instructions.
- Linux users: QT is included in most Linux distributions. In Red Hat 9, it’s in the /usr/lib/qt-x.x directory.
- Windows users: QT does not provide a free Windows version. Please refer to 'User Guide - Batch' to install and use NOMAD without the GUI.

2.3 Cache installation (optional)

NOMAD allows the possibility of storing function values in a cache in order to reduce expensive function evaluations. If the function is cheap, and wall clock time is the measure of efficiency, then the cache may not be needed. If number of function evaluations is the measure of efficiency, then the cache should be used. NOMAD was designed to be used jointly with the Berkeley database. It may be downloaded from this address:

http://www.sleepycat.com/download/index.shtml

Follow BerkeleyDB's installation instructions. For the configure part make sure that you use the '--enable-cxx' option. It will build the C++ API.

2.4 Building NOMAD without the cache

- Copy the ‘gui_nomad.pro’ file from the ‘NOCACHE’ directory to NOMAD’s main directory: ‘cp NOCACHE/gui_nomad.pro .’
- Nomad's Makefile must be generated with 'QMake', QT's Makefile generator (‘QMake’ can be found in QT’s ‘bin’ directory). QMake uses files with the ‘.pro’ extension to generate a Makefile.
  Type 'qmake –o Gui_Makefile gui_nomad.pro'
- The Makefile is generated; type 'make –f Gui_Makefile' to build NOMAD. The executable is called ‘gui_nomad’.

2.5 Building NOMAD with the cache

- First copy these files from the ‘CACHE’ directory to NOMAD’s main directory: gui_nomad.pro, cachecontrol.h, cachecontrol.cpp (it’s ok to write over the ‘cachecontrol’ files):
  ‘cp CACHE/gui_nomad.pro .’
  ‘cp CACHE/cachecontrol.* .’ (Answer ‘yes’ to the overwrite questions).
- NOMAD's Makefile must be generated with 'QMake', QT's Makefile generator (‘QMake’ can be found in QT’s ‘bin’ directory). QMake uses files with the ‘.pro’ extension to generate a Makefile.
  Type 'qmake –o Gui_Makefile gui_nomad.pro'
- The generated Makefile must be modified for the cache to work properly. Open ‘Gui_Makefile’ in a text editor.
  - Before the ‘INCPATH’ variable, write this on a new line:
    BERKELEY_DIR = (the directory where you installed BerkeleyDB)
  - Modify the ‘INCPATH’ variable by adding this string at the end:
    -I$(BERKELEY_DIR)/include
  - Modify the ‘LFLAGS’ variable by adding this string at the end:
    -R$(BERKELEY_DIR)/lib
  - Modify the ‘LIBS’ variable by adding this string in the middle (right after ‘-L$(QTDIR)/lib’):
    -L$(BERKELEY_DIR)/lib -ldb_cxx -ldb
- Save the Makefile and type 'make –f Gui_Makefile' to build NOMAD. The executable is called ‘gui_nomad’.

2.6 Building the test problem

Go in the 'HS23' directory. Compile the black boxes for the objective function and the three general constraints functions, using the g++ compiler:

- g++ –O2 –o truth.exe truth.cpp
- g++ –O2 –o cons1.exe cons1.cpp
- g++ –O2 –o cons2.exe cons2.cpp
- g++ –O2 –o cons3.exe cons3.cpp

3- Command-line processing

NOMAD is run using command-line arguments, in this way:

```
./gui_nomad (description_file) (parameter_file)
```

The description and parameter files are optional. Thus NOMAD can be run three different ways:

- ./gui_nomad : The GUI will appear with no file description loaded and default parameters.
- ./gui_nomad description_file : The GUI will appear with loaded description file and default parameters.
- ./gui_nomad description_file parameter_file : The GUI will appear with loaded description file and loaded parameter file.

4 – How to run the test problem

In this section we will run the test problem included in the download, HS23, with black boxes.

4.1 Test problem: Hock and Schittkowski 23 (HS23)

```
Min f(x) = x1^2 + x2^2
    s.t.    –x1 – x2 + 1 <= 0
            –x1^2 + x2 <= 0
            –x2^2 + x1 <= 0
            –50 <= xi <= 50 , i = 1, 2
```

4.2 Test problem configuration – HS23 with black boxes

Go to the directory where NOMAD is installed. Start the program with the HS23 description and parameter files as command-line arguments, in this way:

```
./gui_nomad HS23/description.dat HS23/parameters.dat
```

The GUI appears on the screen, with both the description and parameter files loaded. The test problem is ready to run, but please take a moment to look at the problem description. Click 'Description->Edit' on the menu bar, it opens a window displaying the problem description. The paths for the various files needed to run the problem are all relative to the directory where NOMAD is installed, so the test problem should run without a glitch.

Don’t change anything, unless the Berkeley database is installed and you want to try out the cache. In this case click on the ‘Use caches?’ checkbox, then on the ‘Enter’ button to the right of the checkbox. A file dialog window will open: navigate the dialog and choose a directory and a file name for the permanent cache (HS23/cache for example). Save and close the file dialog window.

You can also check if the problem files are correctly loaded:

- To view the starting point, click on the ‘View’ button to the right of the ‘Starting point file:’ text. A text editor opens, the starting point should be {3.0, 1.2}. Close the editor.
- To view the bound constraints, click on the ‘View’ button to the right of the ‘Use bounds?’ checkbox. A text editor opens, the lower bounds should be {-50.0, -50.0} and the upper bounds {50.0, 50.0}. Close the editor.
- To view the general constraints’ file names, click on the ‘View’ button to the right of the ‘General constraints:’ text. A text editor opens, the general constraints’ file names should be {./cons1.exe, ./cons2.exe, ./cons3.exe}. Close the editor.

The test problem is ready to be solved.

4.3 Solving the test problem

On the GUI main window, press the 'Solve' button (bottom right). When the run is over, NOMAD will generate these outputs in the window’s right side:

- Best point: Coordinates of the best iterate found during the run.
- Objective function value at best point.
- New truth evaluations: Number of times that the truth black box was called. The truth black box isn’t called when a point was found in the temporary or permanent caches.
- Temporary/permanent cache hits: the temporary cache hit counter is incremented when a point is found in the temporary cache (it means that the point was generated more than once in the current run). The permanent cache hit counter is incremented when a point is found in the permanent cache (it means that the point was also generated in an earlier run). If the cache isn’t installed or isn’t used in the run, ‘0/0’ will be displayed.
- Termination factor: The termination criterion that stopped the run is displayed here.
- Output graph: 'Objective function value' vs. 'Number of function evaluations'.
- Filter graph: Objective function value (f(x)) vs. infeasibility function value (h(x)) appears in a small window.

NOMAD also generates a results file and a history file for each run. Detailed information on the current run can be seen by clicking on the ‘View results’ button. The results file, ‘results.txt’ is located in the HS23 directory. You can compare your results file to the sample results file ‘sample_results.txt’. A history file containing all the points evaluated in the run can be found in the LIBRARY directory. The file is called 'history.txt'. Once the run is over; you have these choices:

- Change the parameters in the main window’s left side and press the 'Continue run' button. This will start a new run, starting from the best iterate found during the previous run.
- Press the 'New run' button, and enter a new problem, or retry the same problem with different strategies.

- Quit the program.

5 – The menu bar

5.1 Click on the ‘Description’ item: These sub-items can be chosen.

- New: An empty description window appears, all the fields are blank. Enter the problem information in the fields according to your problem (see section 6, ‘Description window’). Save and close the window.
- Load: Choose a problem description file in the file dialog. A description window will appear containing your problem information.
- Edit: If a path for a problem description was given, a description window containing your problem information will appear. Edit then save and close the description window.

5.2 Click on the ‘Parameters’ item: These sub-items can be chosen.

- Load: Choose a parameter file in the file dialog. The parameters will be loaded automatically in the left side of the main window.
- Set to defaults: The problem parameters will be changed to defaults.

5.3 Click on the ‘Preferences’ item: This sub-item can be chosen.

- Edit: The preferences window will appear. Edit the preferences as you like and close the window (the preferences are described in chapter 8). Your preferences will be saved automatically.

6 - The description window

The problem description is entered in the description window. This window is opened by accessing the ‘Description’ item on the menu bar. These are the problem description categories, from top to bottom:

6.1 General description:

- Problem name.
- Problem dimension: The number of variables.
- Use caches? Click on this checkbox if you want to use the caches, then on the ‘Enter’ button: select a directory and a file name for the permanent cache by navigating the file dialog.

6.2 Constraints:

- Number of general constraints.
- Use bounds? Click on this checkbox if the problem has bound constraints, then on the ‘Enter’ button: select a directory and a file name where the bound constraints will be stored.
- Click on the ‘View’ button to edit the bound constraints file. Enter all the lower bounds, followed by all the upper bounds: one number per line. You can put comments in the file to make it less confusing: a line is commented if it starts with the '%' character (see the file ‘bounds.txt’ in the ‘HS23’ directory).

6.3 Problem files:

- Directory: Enter the problem directory by clicking on the ‘Enter’ button and navigating the file dialog. This is the directory where your problem files will be stored.
- Starting point: Select a directory and a file name for the starting point file by clicking on the ‘Enter’ button.
- Click on the ‘View’ button to edit the starting point file. Enter the coordinates of your problem’s starting point, one coordinate per line (see the file ‘start_pt.txt’ in the ‘HS23’ directory).
- Results file: Select a directory and a file name for the results file by clicking on the ‘Enter’ button.

6.4 Black boxes: Click on this checkbox if you’re using black boxes, it will activate these items:

- Input file: Select a directory and a file name for the input file by clicking on the ‘Enter’ button.
- Truth executable: Select the directory and file name for the truth executable by clicking on the ‘Enter’ button. The truth executable is the black box for the objective function.
- Use surrogate? Click on this checkbox if the problem uses a surrogate, then on the ‘Enter’ button: select a directory and file name for the surrogate executable.
- General constraints: If the problem has general constraints, select a directory and file name for the general constraint file by clicking on the ‘Enter’ button.
- Click on the ‘View’ button to edit the file. Enter the paths of each general constraint black box, one path per line. If the black boxes are in the problem directory just enter the file names; if they’re elsewhere the full paths are required (see the file ‘gen_cons.txt’ in the ‘HS23’ directory).

6.5 Scaling:

- Scale variables? Select a scaling method by clicking on the combo box. The user has these scaling choices:
    - No scaling: This is the default option.
    - Automatic: NOMAD will handle the scaling automatically.
    - User vector: The user has to provide a scaling vector (see below).

    - User bounds: The user has to provide a lower and upper bound for scaling the variables.
- User vector: Enter a scaling vector by pressing the ‘Enter’ button and navigating the file dialog. The file can be edited by pressing the ‘View’ button. The vector must be of the same length as the problem dimension, one coordinate per line.
- Lower bound: Enter the lower bound used to scale the variables.
- Upper bound: Enter the upper bound used to scale the variables.

6.6 Buttons:

- Save: Press this button when the description file already exists. If it doesn’t exist, this button behaves as the ‘Save as’ button.
- Save as: A file dialog window opens: choose a location and a name for your description file.
- Close: Pressing this closes the description window.

7 – The parameters window

Once a problem is loaded and configured, you can choose the problem parameters on the main window’s left side. There are the parameter categories, from top to bottom:

7.1 Mesh parameters

- Initial poll size: The size of the initial mesh. Default is 1.0.
- Coarsening exponent: When there's an iteration success, the poll size is multiplied by: (poll size basis ^ coarsening exponent). Default is 1.
- Poll size basis: Default is 2.0 for GPS poll directions, 4.0 for MADS poll directions.
- Refining exponent: When there's an iteration failure, the poll size is multiplied by: (poll size basis ^ refining exponent). Default is -1.
- Maximum poll size: The poll size can never go over this value. Default is 100000.

7.2 Poll parameters

- Poll order: There are two strategies for ordering poll directions:
    - Dynamic: The direction that generates a better point is moved to the start of the list.
    - Fixed: The directions are always polled in the same order.
- Poll type: There are two types of poll:
    - Opportunistic: The poll stops as soon as a better point is found.
    - Complete: The poll evaluates all generated points.
- Poll directions: There are five choices of poll directions. As a general rule, the GPS directions are constant throughout the run, whereas the MADS directions are generated anew at every iteration.

    - GPS – 2*n: (2 * dimension) poll directions.
    - GPS – n+1: (dimension + 1) poll directions.
    - GPS – uniform: (dimension + 1) poll directions that are equidistant from one another.
    - MADS – 2*n: (2 * dimension) poll directions that are randomly generated at each iteration.
    - MADS – n+1: (dimension + 1) poll directions that are randomly generated at each iteration.
- Random seed: The random seed is used to generate MADS poll directions and to generate points in the random and Latin hypercube searches.

7.3 Search parameters

The user can choose to do initial, iterative and speculative searches. An initial search is performed at the start of the run, all subsequent searches are iterative.

- Initial search:
    - No search: No initial search will be done.
    - Random: The software will do a random initial search.
    - Latin hypercube: The software will do a Latin hypercube initial search.
- Initial type: There are two types of initial search:
    - Opportunistic: The initial search stops as soon as a better point is found.
    - Complete: The initial search evaluates all generated points.
- Initial points: The number of points that the initial search will generate. Default is (dimension ^ 2).
- Iterative search:
    - No search: No searches will be done in the run, only local polls.
    - Random: The software will do a random search at every iteration.
    - Latin hypercube: The software will do a Latin hypercube search at every iteration.
- Iterative type: There are two types of iterative search:
    - Opportunistic: The iterative search stops as soon as a better point is found.
    - Complete: The iterative search evaluates all generated points.
- Iterative points: The number of points generated by the iterative searches. Default is (dimension * 2).
- Speculative search (MADS): If this box is checked, one point will be added to the search after a successful iteration. This point is generated by making a jump along the last successful poll direction. The speculative search is used only with MADS poll directions: if GPS poll directions are used, checking this box has no effect.

7.4 Termination criteria

The user can choose one or more of the five termination criteria among these five (assign ‘minus one’ to any unwanted criterion):

- Poll size: The run will end when the current poll size is less than or equal to this value.
- Iterations: The run will end after it has completed this number of iterations (a search and a poll phase constitute one iteration).
- Truth evals: Only the function evaluations that do not belong to the temporary cache are counted. A run stops after this number of function evaluations that are either in the permanent cache or in none of the caches.
- New truth evals: Only the function evaluations that do not belong to either cache are counted. A run stops after this number of calls to the user-provided function code is performed.
- Consecutive failures: NOMAD terminates after this number of consecutive function evaluations fail to generate a new incumbent solution.

7.5 Filter parameters

General constraints C(x) <= 0 are handled by NOMAD through a filter. The user may want to handle linear or even bound constraints with the filter. The box must be checked if the problem has general constraints.

- Hmax: Hmax can be relative to the starting point's h(x) value, or have a fixed value.
- Hmax value: If Hmax is relative, it's calculated as follow:
  (starting point's h(x) * Hmax value)
  If Hmax is fixed, it's this value.
- Filter norm: Click on the combo box and choose between 'L-2 squared' (default), 'L-Infinity' or 'L-1' filter norm.
- Hmin value: When a point's h(x) value is under 'Hmin', it's considered feasible and its h(x) value is put to zero.

7.6 Surrogate parameters: The box must be checked if the problem uses a surrogate.

- Surrogate tolerance: When a trial point's surrogate function value is greater than the incumbent's surrogate function value by this factor, NOMAD stops evaluating Truth for this list of points (this is safe because the points are ordered by surrogate function value).

7.7 Buttons

- Save: Press this button when the parameter file already exists. If it doesn't exist, this button behaves as the 'Save as' button.
- Save as: A file dialog window opens: choose a location and a name for the parameter file.

8 – The preferences window

This window contains the user’s preferences. It is opened by accessing the ‘Preferences’ item on the menu bar. The user’s preferences are:

- Display factor: This variable determines how much information will be displayed in the terminal where NOMAD was started. It goes from ‘0’ (no information) to ‘10’ (the most information). It’s useful if the user wants to see what the program does, what points are generated, etc.
- Send email after run is over? Check this box if you want to be alerted by email at the end of a run. This is useful for long runs.
- Email address: Enter your email address in the field.

Press the ‘Close’ button to close the window. Your preferences will be saved in the ‘LIBRARY/preferences.txt’ file.

9 – How to solve your own optimization problem

9.1 With black boxes (separately compiled executables that read an input file containing values of x and print out the value of the function evaluation at x)

Make a directory where you'll store your black boxes, input and result files (as in the HS23 problem). Build separate executables for the objective, surrogate and general constraint functions and put them in this directory.

The point of using separate executables is to save computation. For example, if any bound constraint is violated, then the point is unacceptable without any further computation. Also in the filter method, when the constraints violate the maximum allowed constraint violation, then there is no need to evaluate any remaining constraints or the objective function.

9.2 Without black boxes

If you're not using black boxes, you can write your objective and general constraint functions directly inside NOMAD.

- Write your own objective function in 'TruthFunction::evaluate(…)' (file 'truthfunction.cpp').
- Write your own general constraint function in 'GeneralConstraints::userRoutine(...)' (file 'generalconstraints.cpp').
- Save the modified files and recompile Nomad.

9.3 Create a new description file

Start the program and create a new problem file by clicking 'Description->New' on the menu bar. An empty description window appears, all the fields are blank. Enter the problem information in the fields according to your problem (see section 6, ‘The description window’). Click on the 'Save as' button. A file dialog window opens: choose a location and a name for your description file. Close the window.

9.4 Choose problem parameters

Choose the parameters you want for the problem by filling the left side of the main window (see chapter 7, ‘The parameters window’).

NOMAD is ready to run your problem. Click on the ‘Solve’ button.