User guide to NOMAD without 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 nonlinear nonsmooth 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 fails 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 BerkeleyDB 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 BerkeleyDB database.

2.2 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 wallclock 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 BerkeleyDB cache. 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.

2.3 Building NOMAD without the cache

- Type 'make –f Batch_Makefile' to build NOMAD. The executable is called 'batch_nomad'.

2.4 Building NOMAD with the cache

- First copy these files from the 'CACHE' directory to NOMAD's main directory: cachecontrol.h, cachecontrol.cpp (it's ok to write over the 'cachecontrol' files):
        'cp CACHE/cachecontrol.* .' (Answer 'yes' to the overwrite questions).
- Open 'Batch_Makefile' in a text editor. Change the variable 'BERKELEY_DIR' (line 5) to the directory where you installed BerkeleyDB.
- Save 'Batch_Makefile' and type 'make –f Batch_Makefile' to build NOMAD. The executable is called 'batch_nomad'.

2.5 Building the test problem

Go in the 'PROBLEMS/HS23' directory. Compile the black boxes for the objective function and the three general constraints functions, using either the CC or 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:

./batch_nomad description_file (parameter_file)

The parameter file is optional. Thus NOMAD can be run two different ways:

- ./batch_nomad description_file : The program will solve the loaded problem description with default parameters.
- ./batch_nomad description_file parameter_file : The program will solve the loaded problem description with parameters loaded from the parameters file.

4 – How to run the test problem

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

4.1 Test problem: Hock and Schittkowski 23 (HS23)

$$\text{Min } f(x) = x_1^2 + x_2^2$$
$$\text{s.t.} \quad -x_1 - x_2 + 1 \le 0$$
$$-x_1^2 + x_2 \le 0$$
$$-x_2^2 + x_1 \le 0$$
$$-50 \le x_i \le 50 , i = 1, 2$$

4.2 Test problem configuration – HS23 with black boxes

The test problem is already configured and ready to run, but please take a moment to look at the problem description and parameters.

Open the file 'HS23/description.dat' in a text editor. This file contains all the information defining the HS23 problem: dimension, constraints, problem files, black boxes, scaling. Chapter 5 describes the description file in detail.

Open the file 'HS23/parameters.dat' in a text editor. This file contains the problem parameters for the mesh, poll, search, termination criteria and filter. Chapter 6 describes the parameters file in detail.

4.3 Solving the test problem

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:

./batch_nomad HS23/description.dat HS23/parameters.dat

When the run ends Nomad will produce two outputs:
- Detailed information on the run can be found in the results file, 'HS23/results.txt'. You can compare your test results to the sample results file, 'HS23/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'.

5 - The description file

Every problem must be defined in a description file ('HS23/description.dat' is the description of the HS23 problem). Every line in the file is composed of a variable name in capital letters, followed by a value for that particular variable. Some variables can be omitted if they're not used in the problem (a lot of string variables are defined only if the corresponding boolean variable is 'true' for example). The list of variables, their type and value range is:

- PROBLEM_NAME [string]: The name given to this problem.

- DIMENSION [integer]: The number of optimization variables.

- USE_CACHES [boolean]: Are the caches used with this problem?
    - 0: Don't use the caches.
    - 1: Use the caches (the variable 'CACHE_FILE' must be defined).

- CACHE_FILE [string]: The path of the permanent cache.

- GEN_CONS_NB [integer]: The number of general constraints.

- USE_BOUNDS [boolean]: Are there bound constraints in this problem?
    - 0: No bound constraints.
    - 1: Bound constraints (the variable 'BOUNDS_FILE' must be defined).

- BOUNDS_FILE [string]: The file where bound constraints are stored.

- DIRECTORY [string]: The directory where the problem files are stored.

- START_PT_FILE [string]: The file where the starting point is stored.

- RESULTS_FILE [string]: The file where the problem results will be written.

- USE_BLACK_BOXES [boolean]: Does the problem use black boxes? (separately compiled executables for the objective function and general constraints)
    - 0: Black boxes are not used.

- 1: Black boxes are used (the variables 'INPUT_FILE' and 'TRUTH_EXE' must be defined; 'GEN_CONS_FILE' must also be defined if there are general constraints in the problem).

- INPUT_FILE [string]: The points evaluated by the black boxes will be written in the input file.

- TRUTH_EXE [string]: A path for the objective function black box.

- GEN_CONS_FILE [string]: This file contains the paths of all the general constraint black boxes. See the file 'HS23/gen_cons.txt'.

- USE_SURROGATE [boolean]: Does the problem use a surrogate?
  - 0: No surrogate.
  - 1: A surrogate function is used (the variable 'SURROGATE_EXE' must be defined if black boxes are in use).

- SURROGATE_EXE [string]: A path for the surrogate function black box.

- SCALING_CHOICE [integer]: The scaling method for this problem. These are the four scaling methods in NOMAD:
  - 0: No scaling.
  - 1: Automatic scaling.
  - 2: User vector (the variable 'SCALES_FILE' must be defined).
  - 3: User bounds (the variables 'LOWER_SCALE' and 'UPPER_SCALE' must be defined).

- SCALES_FILE [string]: This file contains a vector used to scale the variables.

- LOWER_SCALE [real]: A lower bound used to scale the variables.

- UPPER_SCALE [real]: An upper bound used to scale the variables.

6 – The parameter file

NOMAD can optionally be run with a parameter file (see chapter 3 'Command-line processing'). Every line in the file is composed of a variable name in capital letters, followed by a value for that particular variable. Some variables can be omitted if they're not used in the problem. This is the list of variables, their type and value range by parameter category:

6.1 Mesh parameters

- INITIAL_POLL_SIZE [real]: The size of the initial poll size. Default is 1.0.
- MAX_POLL_SIZE [real]: Upper bound for the poll size. Default is 100000.

- POLL_BASIS [real]: Default is 2.0 for GPS poll directions, 4.0 for MADS poll directions.
- COARSENING_EXPONENT [integer]: When there's an iteration success, the poll size is multiplied by: (poll basis ^ coarsening exponent). Default is 1.
- REFINING_EXPONENT [integer]: When there's an iteration failure, the poll size is multiplied by: (poll basis ^ refining exponent). Default is -1.

6.2 Poll parameters ('n' represents the problem dimension)

- POLL_ORDER [boolean]: There are two strategies for ordering poll directions:
    - 0: Dynamic directions. The direction that generates a better point is moved to the start of the list.
    - 1: Fixed directions. The directions are always polled in the same order.
- POLL_COMPLETE [boolean]: Is the poll complete?
    - 0: Opportunistic poll. The poll stops as soon as a better point is found.
    - 1: Complete poll. The poll evaluates all generated points.
- POLL_DIRECTIONS [integer]: 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.
    - 0: GPS 2*n poll directions.
    - 1: GPS n+1 poll directions.
    - 2: GPS uniform: The n+1 poll directions are equidistant from one another.
    - 3: MADS 2*n poll directions that are randomly generated at each iteration.
    - 4: MADS n+1 poll directions that are randomly generated at each iteration.
- RANDOM_SEED [integer]: The random seed is used to generate MADS poll directions and to generate points in the random and Latin hypercube searches.

6.3 Search parameters ('n' represents the problem dimension)

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 [integer]:
    - 0: No initial search.
    - 1: Random initial search.
    - 2: Latin hypercube initial search.
- INITIAL_COMPLETE [boolean]: Is the initial search complete?
    - 0: Opportunistic search. The initial search stops as soon as a better point is found.
    - 1: Complete search. The initial search evaluates all generated points.
- INITIAL_POINTS [integer]: The number of points generated by the initial search. Default is (dimension ^ 2).
- ITERATIVE_SEARCH [integer]:
    - 0: No iterative search.
    - 1: Random iterative search.

- o 2: Latin hypercube iterative hypercube search.
- ITERATIVE_COMPLETE [boolean]: Is the iterative search complete?
    - o 0: Opportunistic search. The iterative search stops as soon as a better point is found.
    - o 1: Complete search. The iterative search evaluates all generated points.
- ITERATIVE_POINTS [integer]: The number of points generated by the iterative search. Default is (dimension * 2).
- SPECULATIVE_SEARCH [boolean]: If this box is checked, one point will be added to the search after a successful iteration. This point is generated by jumping ahead 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.

6.4 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_TERM [real]: The run will end when the current poll size is less than or equal to this value.
- ITERATIONS [integer]: The run will end after it has completed this number of iterations (a search and a poll phase constitute one iteration).
- TRUTH_EVALS [integer]: 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 [integer]: 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_FAILS [integer]: NOMAD terminates after this number of consecutive function evaluations fail to generate a new incumbent solution.

6.5 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.

- FILTER_RELATIVE [boolean]:
    - o 0: Hmax is fixed.
    - o 1: Hmax is relative to the starting point's h(x) value.
- HMAX [real]: Upper bound on filter points' h(x) values. Points whose h(x) values are above Hmax are considered too infeasible and are not included in the filter. If 'FILTER_RELATIVE' is true, Hmax is calculated as follow:
    (Starting point's h(x) * HMAX)
    If Hmax is fixed, it's equal to this value.

- HMIN [real]: Lower bound on filter points' h(x) values. Points whose h(x) values are under Hmin are considered feasible and their h(x) values are put to zero.
- FILTER_NORM [integer]:
  - 0: L-Infinity norm.
  - 1: L-1 norm.
  - 2: L-2 squared norm (default).

6.6 Surrogate parameters

- SURROGATE_TOLERANCE [real]: 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 – The preferences file

The user's preferences can be saved in the 'LIBRARY/preferences.txt' file. Every line in the file is composed of a variable name in capital letters, followed by a value for that particular variable. This is the list of variables, their type and value range:

- DISPLAY_FACTOR [integer]: 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 [boolean]: Does the user want to receive an email when the run is over? (This is useful for long runs.)
  - 0: Don't send an email.
  - 1: Send an email (the variable 'EMAIL_ADDRESS' must be defined).
- EMAIL_ADDRESS [string]: The user's email address.

8 – How to solve your own optimization problem

- With black boxes:
  - Create 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 the directory you just created.
- Without black boxes: The problem's objective and general constraint functions have to be written in NOMAD's code.
  - 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.

- Create a description file (see Chapter 5 for reference) by copying the file 'HS23/description.dat' and modifying it.
- Create a parameter file (see Chapter 6 for reference) by copying the file 'HS23/parameters.dat' and modifying it.
- Solve the problem by typing this in a terminal (see Chapter 3 for reference):
    ./batch_nomad description_file parameter_file