Real-Parameter Black-Box Optimization Benchmarking 2009: Experimental Setup

Nikolaus Hansen, Anne Auger, Steffen Finck, Raymond Ros

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Real-Parameter Black-Box Optimization Benchmarking 2009: Experimental Setup

Nikolaus Hansen — Anne Auger — Steffen Finck — Raymond Ros

N° 6828

February 2009

Thème COG

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--6828--FR+ENG

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

centre de recherche SACLAY – ÎLE-DE-FRANCE

Real-Parameter Black-Box Optimization Benchmarking 2009: Experimental Setup

Nikolaus Hansen[*], Anne Auger[†], Steffen Finck[‡], Raymond Ros[§]

Thème COG — Systèmes cognitifs
Équipes-Projets Adaptive Combinatorial Search et TAO

Rapport de recherche n° 6828 — February 2009 — 15 pages

Abstract: Quantifying and comparing performance of optimization algorithms is one important aspect of research in search and optimization. However, this task turns out to be tedious and difficult to realize even in the single-objective case – at least if one is willing to accomplish it in a scientifically decent and rigorous way. The BBOB 2009 workshop will furnish most of this tedious task for its participants: (1) choice and implementation of a well-motivated single-objective benchmark function testbed, (2) design of an experimental set-up, (3) generation of data output for (4) post-processing and presentation of the results in graphs and tables. What remains to be done for the participants is to allocate CPU-time, run their favorite black-box real-parameter optimizer in a few dimensions a few hundreds of times and execute the provided post-processing script afterwards. Here, the experimental procedure and data formats are thoroughly defined and motivated and the data presentation is touched on.

Key-words: optimization, evolutionary algorithms, benchmarking, black-box

[*] NH is with the Microsoft Research–INRIA Joint Centre, 28 rue Jean Rostand, 91893 Orsay Cedex, France
[†] AA is with the TAO Team, INRIA Saclay, Université Paris Sud, LRI, 91405 Orsay cedex, France
[‡] SF is with the Research Center PPE, University of Applied Sciene Vorarlberg, Hochschulstrasse 1, 6850 Dornbirn, Austria
[§] RR is with the Univ. Paris-Sud, LRI, UMR 8623 / INRIA Saclay, projet TAO, F-91405 Orsay, France.

Centre de recherche INRIA Saclay – Île-de-France
Parc Orsay Université
4, rue Jacques Monod, 91893 ORSAY Cedex
Téléphone : +33 1 72 92 59 00

No French Title

Résumé : Pas de résumé

Mots-clés : Pas de motclef

Contents

1 Introduction

This document presents the experimental setup and the data presentation for BBOB. The definition of the benchmark functions [4][5] and a the technical documentation for the provided software is given elsewhere.

1.1 Symbols, Constants, and Parameters

f_{opt} optimal function value is defined for each benchmark function individually

f_{target} target function value to reach. The final, smallest considered target function value is $f_{\mathrm{target}} = f_{\mathrm{opt}} + 10^{-8}$, but also larger values for f_{target} are evaluated.

`Ntrials` $= 15$ is the number of trials for each single setup, i.e. each function and dimensionality

$D = 2; 3; 5; 10; 20; 40$ search space dimensionalities used for all functions. Dimensionality 40 is optional and can be omitted.

2 Benchmarking Experiment

The real-parameter search algorithm under consideration is run on a testbed of benchmark functions to be minimized. Each run is repeated `Ntrials` times for each function and each dimensionality. In each trial a *different instance* of the function is generated (instantiations $1, 2, \ldots,$ `Ntrials` are used). A MATLAB example script for this procedure is given in Figure 1. The algorithm is run on *all* functions of the testbed under consideration.

2.1 Input to the Algorithm and Initialization

An algorithm can use the following input.

1. the search space dimensionality D
2. the search domain; all functions are defined everywhere in $\mathcal{R}^D$ and have their global optimum in $[-5, 5]^D$. Most functions have their global optimum in $[-4, 4]^D$ which can also be a reasonable setting for initial solutions.
3. indication of the testbed under consideration, i.e. different algorithms and/or parameter settings might well be used for the noise-free and the noisy testbed
4. the target function value f_{target} is provided for conclusive termination of trials, in order to reduce the overall CPU requirements. The target function value is not intended to be utilized as algorithm input otherwise.

Based on these input parameters, the parameter setting and initialization of the algorithm is entirely left to the participants. As a consequence, the setting shall be identical for all benchmark functions of one testbed (the function identifier or any known characteristics of the function are not meant to be input to the algorithm, see also Section 4).

2.2 Termination Criteria and Restarts

Each trial can be conclusively terminated if f_{target} is reached. Otherwise, the choice of termination is considered as part of the algorithm. Algorithms with any budget of function evaluations, small or large, will be considered in the analysis of the results. Reasonable termination criteria are recommended, but exploiting a larger number of function evaluations might increase the chance to achieve better function values or solve the function up to the final f_{target}[1]. We suggest to consider a multistart procedure.

Independent restarts are the most simple meta-heuristic for multistarts. For example, given a fast algorithm with a small success probability, say 5% (or 1%), chances are that not a single trial (out of 15) will be successful. With 10

[1]We expect that the easiest functions can be solved in less than $10D$ function evaluations, while the most difficult functions might need a budget of more than $1000D^2$ function evaluations to reach *the final* $f_{\text{target}} = f_{\text{opt}} + 10^{-8}$.

Figure 1: exampleexperiment.m: example for benchmarking MY_OPTIMIZER on the noise-free function testbed in MATLAB/Octave. An example for the function MY_OPTIMIZER is given in Appendix A

```
% runs an entire experiment for benchmarking MY_OPTIMIZER
% on the noise-free testbed. fgeneric.m and benchmarks.m
% must be in the path of Matlab/Octave
% CAPITALIZATION indicates code adaptations to be made

addpath('PUT_PATH_TO_BBOB/matlab');  % should point to fgeneric.m etc.
datapath = 'PUT_MY_BBOB_DATA_PATH';  % different folder for each experiment
opt.algName = 'PUT ALGORITHM NAME';
opt.comments = 'PUT MORE DETAILED INFORMATION, PARAMETER SETTINGS ETC';

more off;  % in octave pagination is on by default

t0 = clock;
rand('state', sum(100 * t0));

for dim = [2,3,5,10,20,40]  % small dimensions first, for CPU reasons
  for ifun = benchmarks('FunctionIndices')
    for itrial = 1:15  % first 15 function instances
      fgeneric('initialize', ifun, itrial, datapath, opt);  % instance itrial

      MY_OPTIMIZER(@fgeneric, dim, fgeneric('ftarget'), inf);

      disp(sprintf(['  f%d in %d-D, trial %d: FEs=%d,' ...
                    ' fbest-ftarget=%.4e, elapsed time [h]: %.2f'], ...
                   ifun, dim, itrial, ...
                   fgeneric('evaluations'), ...
                   fgeneric('fbest') - fgeneric('ftarget'), ...
                   etime(clock, t0)/60/60));
      fgeneric('finalize');
    end
    disp(['      date and time: ' num2str(clock, ' %.0f')]);
  end
  disp(sprintf('---- dimension %d-D done ----', dim));
end
```

(or 90) independent restarts, the success probability will increase to 40% and the performance of the algorithm will become visible. At least 4–5 successful trials are desirable to accomplish a stable performance measurement. Generally, independent restarts do not change the main performance measure ERT (see Appendix C.2), they only improve the reliability of the measured value. This reasoning is valid for any possible target function value under consideration.

Restarts with different parameter setups, for example with different (increasing) population sizes, might be considered as well, as it has been applied quite successfully [2]. Choosing different setups mimics what will be done in practice.

Figure 2: exampletiming.m: example for measuring the time complexity of MY_OPTIMIZER given in MATLAB/Octave. An example for MY_OPTIMIZER is given in Appendix A

```
% runs the timing experiment for MY_OPTIMIZER. fgeneric.m
% and benchmarks.m must be in the path of MATLAB/Octave

addpath('PUT_PATH_TO_BBOB/matlab');  % should point to fgeneric.m etc.

more off;  % in octave pagination is on by default

timings = [];
runs = [];
dims = [];
for dim = [2,3,5,10,20,40]
  nbrun = 0;
  ftarget = fgeneric('initialize', 8, 1, 'tmp');
  tic;
  while toc < 30  % at least 30 seconds
    MY_OPTIMIZER(@fgeneric, dim, ftarget, inf);
    nbrun = nbrun + 1;
  end  % while
  timings(end+1) = toc / fgeneric('evaluations');
  dims(end+1) = dim;    % not really needed
  runs(end+1) = nbrun;  % not really needed
  fgeneric('finalize');
  disp([['Dimensions:' sprintf(' %11d ', dims)]; ...
        ['      runs:' sprintf(' %11d ', runs)]; ...
        [' times [s]:' sprintf(' %11.1e ', timings)]]);
end
```

All restart mechanisms are finally considered as part of the algorithm under consideration.

3 Time Complexity Experiment

In order to get a rough measurement of the time complexity of the algorithm, the overall CPU time is measured when running the algorithm on f_8 (Rosenbrock function) for at least a few tens of seconds (and at least a few iterations). The chosen setup should reflect a "realistic average scenario". If another termination criterion is reached, the algorithm is restarted (like for a new trial). The *CPU-time per function evaluation* is reported for each dimension. The time complexity experiment is conducted in the same dimensions as the benchmarking experiment. The chosen setup, coding language, compiler and computational architecture for conducting these experiments are described. Figure 2 shows a respective MATLAB/Octave code example. For CPU-inexpensive algorithms the timing might mainly reflect the time spent in function fgeneric.

4 Parameter setting and tuning of algorithms

The algorithm and the used parameter setting for the algorithm should be described thoroughly. Whether or not all functions were approached with the very same parameter setting (which might well depend on the dimensionality, see Section 2.1) should be stated clearly and the *crafting effort* should be given (see below). The crafting effort is zero, if the setting was identical for all functions. The method of choosing the parameters for the testbed should be described, as well as which parameters were adjusted to the testbed, the number of overall settings evaluated and the number of settings finally used.

In general, we discourage the *a priori* use of function-dependent parameter settings. In other words, we do not consider the function ID or any function characteristics (like separability, multi-modality, ...) as input parameter to the algorithm (see also Section 2.1). Instead, we encourage either using multiple runs with different parameters (for example restarts, see also Section 2.2), or using (other) probing techniques for identifying function-wise appropriate parameters online. The underlying assumption in this experimental setup is that also in practice we do not know in advance whether the algorithm will face f_1 or f_2, a unimodal or a multimodal function, or... and we cannot adjust algorithm parameters *a priori*[2].

In case that, nevertheless, in one dimension $K > 1$ *different* parameter setting were finally used, the following entropy measure (*crafting effort*, see also [3, 6][3]) needs to be provided for each dimensionality D:

$$\mathrm{CrE} = -\sum_{k=1}^{K} \frac{n_k}{n} \ln\left(\frac{n_k}{n}\right) \tag{1}$$

where $n = \sum_{k=1}^{K} n_k$ is the number of functions in the testbed and n_k is the number of functions, where the parameter setting with index k was used, for $k = 1, \ldots, K$. When a single parameter setting were used for all functions, as recommended, the crafting effort is $\mathrm{CrE} = \sum_{k=1}^{1} \frac{n}{n} \ln\left(\frac{n}{n}\right) = 0$.[4]

5 Data to provide

The provided implementations of the benchmark functions generate data for reporting and analysis. Since one goal is the comparison of different algorithms, the data from the experiments shall be submitted to `http://coco.gforge.`

[2]In contrast to most other function properties, the property of having noise can usually be verified easily. Therefore, for noisy functions a *second* testbed has been defined. The two testbeds can be approached *a priori* with different parameter settings or different algorithms.

[3]Our definition differs from [3, 6] in that it is independent of the number of adjusted parameters. Only the number of used different settings is relevant.

[4]We give another example: in 5-D all functions were optimized with the same parameter setting. In 10-D the first 14 functions were approached with one parameter setting and the remaining 10 functions with a second one (no matter how many parameters were changed). In 20-D the first 10 functions were optimized with one parameter setting, functions 11–13 and functions 23–24 were optimized with a second setting, and the remaining 9 functions 14–22 were optimized with a third setting. The crafting effort computes independently for each dimension in 5-D to $\mathrm{CrE}_5 = 0$, in 10-D to $\mathrm{CrE}_{10} = -\left(\frac{14}{24}\ln\frac{14}{24} + \frac{10}{24}\ln\frac{10}{24}\right) \approx 0.679$, and in 20-$D$ to $\mathrm{CrE}_{20} = -\left(\frac{10}{24}\ln\frac{10}{24} + \frac{5}{24}\ln\frac{5}{24} + \frac{9}{24}\ln\frac{9}{24}\right) \approx 1.06$.

`inria.fr/doku.php?id=bbob-2009`. All submitted data will become available to the participants.

6 Post-Processing and Data Presentation

Python scripts are provided to produce tables and figures reporting the outcome of the benchmarking experiment.

Given the output data from the experiment are in the folder `my_data` of the current directory, the following command line needs to be executed[5]:

```
python path_to_postproc_code_folder/bbob_pproc/run.py my_data
```

This will create a folder `ppdata` in the working directory that will contain the output from the post-processing. Finally the command

```
pdflatex templateBBOBarticle.tex
```

executed in the same directory will compile a report template with tables and figures created from the data in `ppdata`. The folder `ppdata`, the files `templateBBOBarticle.tex` and `acm_proc_article-sp.cls` (ACM SIG proceedings template) have to be in the working directory. For the noisy testbed the template `templateBBOBnoisyarticle.tex` is provided.

Acknowledgments

The authors would like to thank Petr Pošík and Arnold Neumaier for the inspiring and helpful discussion.

References

[1] A. Auger and N. Hansen. Performance evaluation of an advanced local search evolutionary algorithm. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2005)*, pages 1777–1784, 2005.

[2] A Auger and N Hansen. A restart CMA evolution strategy with increasing population size. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2005)*, pages 1769–1776. IEEE Press, 2005.

[3] Vitaliy Feoktistov. *Differential Evolution: In Search of Solutions (Springer Optimization and Its Applications)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[4] S. Finck, N. Hansen, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2009: Presentation of the noiseless functions. Technical Report 2009/01, Research Center PPE, 2009.

[5] N. Hansen, S. Finck, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. Technical Report RR-6829, INRIA, 2009.

[5]Python 2.5.x, Numpy, and Matplotlib must be installed. Under Windows the path separator '\' instead of '/' must be used in the command line.

[6] Kenneth Price. Differential evolution vs. the functions of the second ICEO. In *Proceedings of the IEEE International Congress on Evolutionary Computation*, pages 153–157, 1997.

APPENDIX

A Example Optimizer with Multistarts

The optimizer used in Fig. 1 and 2 is given in Fig. 3.

B Rationales Behind the Parameter Settings

Rationale for the choice of `Ntrials` Parameter `Ntrials` determines the minimal measurable success rate and influences the overall necessary CPU time. Compared to a typical standard setup, we have chosen a comparatively small value for `Ntrials`. Consequently, within the same CPU-time budget, single trials can conduct more function evaluations, if needed or desired (only needed as long as f_{target} is not reached). For algorithms with a small success probability, say smaller than 20%, this setup leaves the burden to implement an automated multistart procedure, as it will be necessary in practice (see also Section 2.2).

Rationale for the choice of f_{target} The initial search domain and the target function value are an essential part of the benchmark function definition. Different target function values might lead to a different characteristics of the problem to be solved, besides that larger target values are invariably less difficult to reach. Functions might be easy to solve up to a function value of 1 and become intricate for smaller target values. The actually chosen value for the final f_{target} is somewhat arbitrary and could be changed by simple modifications in the function definition. The performance evaluation will consider a wide range of different target function values to reach, all being larger or equal to the final f_{target}.

C Rationale behind the data presentation

C.1 Performance Measures

We advocate performance measures that are

- quantitative, ideally with a ratio scale (rather than interval or ordinal scale) and with a wide variation (i.e. for example not with values usually ranging between 0.98 and 1)
- well-interpretable, having a semantics
- as simple as possible
- relevant with respect to the "real world"

C.2 Expected Running Time

We use the *expected running time* (ERT) as most prominent performance measure, more precisely, the expected number of function evaluations to reach a target function value for the first time, computed as [1]

Figure 3: Example optimizer used in Fig. 1 and 2

```
function [x, ilaunch] = MY_OPTIMIZER(FUN, DIM, ftarget, maxfunevals)
% minimizes FUN in DIM dimensions by multistarts of fminsearch.
% ftarget and maxfunevals are additional external termination conditions.
% fminsearch was modified to take as input a delta_i to generate
% the first simplex.

  % set options, make sure we always terminate
  options = optimset('MaxFunEvals', min(1e4*DIM, maxfunevals), ...
                     'MaxIter', 2e3*DIM, ...
                     'Tolfun', 1e-11, ...
                     'TolX', 0, ...
                     'OutputFcn', @callback, ...
                     'Display', 'off');

  % multistart such that ftarget is reached with reasonable prob.
  for ilaunch = 1:100;  % relaunch optimizer up to 100 times
    % set initial conditions
    if mod(ilaunch-1, floor(1 + 3 * rand(1,1))) == 0
      xstart = 8 * rand(DIM, 1) - 4;  % random start solution
      delta_i = 4;
    else
      xstart = x;  % try to improve found solution
      delta_i = 0.1;
    end
    % try fminsearch from Matlab, modified to take delta_i as arg
    x = fminsearch_mod(FUN, xstart, delta_i, options);
    if feval(FUN, 'fbest') < ftarget || ...
          feval(FUN, 'evaluations') >= maxfunevals
      break;
    end
    % if useful, modify more options here for next launch
  end

  function stop = callback(x, optimValues, state)
    stop = false;
    if optimValues.fval < ftarget
      stop = true;
    end
  end

end
```

$$\begin{aligned}
\mathrm{ERT}(f_{\mathrm{target}}) &= \mathrm{RT_S} + \frac{1-p_{\mathrm{s}}}{p_{\mathrm{s}}}\mathrm{RT_{US}} && (2)\\
&= \frac{p_{\mathrm{s}}\mathrm{RT_S} + (1-p_{\mathrm{s}})\mathrm{RT_{US}}}{p_{\mathrm{s}}} && (3)\\
&= \frac{\#\mathrm{FEs}(f_{\mathrm{best}} \geq f_{\mathrm{target}})}{\#\mathrm{succ}} && (4)
\end{aligned}$$

where the *running times* $\mathrm{RT_S}$ and $\mathrm{RT_{US}}$ denote the average number of function evaluations for successful and unsuccessful trials, respectively, and p_s denotes the fraction of successful trials, where successful trials are those that reached f_target. The $\#\mathrm{FEs}(f_\mathrm{best} \geq f_\mathrm{target})$ is the number of function evaluations conducted in all trials, while the best function value was not smaller than f_target during the trial, i.e. the sum over all trials of

$$\max\{\mathrm{FE\ s.t\ } f_\mathrm{best}(\mathrm{FE}) \geq f_\mathrm{target}\} \ .$$

The #succ denotes the number of successful trials. ERT estimates the expected running time to reach f_target [1]. ERT is a function of f_target, as in particular $\mathrm{RT_S}$ and p_s depend on the chosen value for f_target.

C.3 Vertical versus horizontal view

There exist two different approaches for collecting data of different runs.

Vertical-cut view. Fixing a number of function evaluations (this corresponds to fixing a cost) and measuring the functions values reached for this given number of function evaluations. Fixing a number of function evaluations can be pictured as drawing a vertical line on the convergence graphs.

Horizontal-cut view. Fixing a target function value (i.e. drawing an horizontal line in the convergence graphs) and measuring the number of function evaluations needed to reach this target function value.

It is often argued that the first approach is close to what is needed for real word applications where the total number of function evaluations is limited. On the other hand, also a minimum target requirement needs to be achieved in real world applications, for example, getting (noticeably) better than the currently available best solution or than a competitor. For benchmarking algorithms we prefer the horizontal cut over the vertical cut since it gives quantitative *and* interpretable data: the horizontal approach is quantitative since it measures the time needed to reach a given target function value and allows therefore to give conclusions of the type: Algorithm A is two/ten/hundred times faster than Algorithm B in solving this problem (i.e. reaching the given target function value). The vertical cut does not give quantitatively *interpretable* data: there is no interpretable meaning to the fact that Algorithm A reaches a fitness value that is two/ten/hundred times smaller than the one reached by Algorithm B. Furthermore, for algorithms invariant with respect to order-preserving transformations of the function value (which is the case of algorithms based on comparisons like DE, PSO, ES), measures based on the horizontal cut will be invariant to these transformations (given a transformed target function value) contrarily to measures based on the vertical cut.

D Data and File Formats

D.1 Introduction

This section specifies the format for the output data files and the content of the files, as they are written by the provided benchmark functions implementations.

```
↪ container_folder
        ↪ fileprefix_f1.info
        ↪ data_f1
          ↪ fileprefix_f1_DIM5.dat
          ↪ fileprefix_f1_DIM5.tdat
          ↪ fileprefix_f1_DIM10.dat
          ↪ fileprefix_f1_DIM10.tdat
        ↪ fileprefix_f2.info
        ↪ data_f2
          ↪ fileprefix_f2_DIM5.dat
          ↪ fileprefix_f2_DIM5.tdat
↪ container_folder2
↪ ...
```

Figure 4: Example data file structures obtained with the BBOB experiment software.

The goal is to obtain format-identical files which can be analyzed with the provided post-processing tools. The first section explains the general settings. Afterwards the format for the different output files will be given in detail and with examples.

D.2 General Settings

The output from one *experiment*, consisting of `Ntrials` optimization runs on a given objective function, will be contained in a folder which path will be decided by the user and will consist of one index file (or more) and two data files (or more). The output files contain all necessary data for post-processing. The extensions are '*.info' for the index file and '*.dat', '*.tdat' for the data files. An example of the folder/file structure can be found in Fig 4. After performing all simulations, the user can use the data files with the provided post-processing tool to obtain LaTeX files, including tables and figures of the results.

D.3 Output Files

D.3.1 Index File

The index file contains meta information on the optimization runs and the location of the corresponding data files. The user is free to choose any prefix for the index file name. The function identifier will be appended to it and the extension will be '.info'. The contents of the index file are the concatenation of 3-line index entries (output format is specified in brackets):

- 1*st* line - function identifier (%d), search space dimension (%d), precision to reach (%4.3e) and the identifier of the used algorithm (%s)

- *2nd* line - comments of the user (e.g. important parameter or used internal methods)
- *3rd* line - relative location and name of data file(s) followed by a colon and information on a single run: the instance of the test function, final number of function evaluations, a vertical bar and the final best function value minus target function value.

All entries in the *1st* line and the *3rd* line are separated by commas.

For each experiment the provided data-writing tools generate one index file with the respective entries. All index files have to be included in the archive for the submission and will contain the *relative* location of the data files *within* the archive. Thus, it is necessary to archive all files in the same folder-subfolder structure as obtained. An example of an index file is given in Fig 5. An entry of

```
funcId = 12, DIM = 5, Precision = 1.000e-08, algId = 'ALG-A'
% parameterA = 2, parameterB = 3.34, ...
data_f12\test_f12_DIM5.dat, 1:387|-2.9e-009, 2:450|-2.8e-009, 3:422|-2.1e-009, data_f12\test-01_f12_DIM5.dat, 1:5000000|1.8e-008,
...
funcId = 12, DIM = 10, Precision = 1.000e-08, algId = 'ALG-A'
% parameterA = 2, parameterB = 3.34, ...
data_f12\test1_f12_DIM10.dat, 1:307|-8.6e-008, 2:321|-3.5e-008, ...
...
```

Figure 5: Example of an index file

the index file is written at the start of the first sample run for a given function and dimension.

D.3.2 Data Files

A data file contains the numerical output of an optimization run on a given objective function. The content of the data file is given in the following. Data files will be placed in subfolders at the location of their corresponding index file. At the start of each sample run the header for the data file is written. The header is one line with the titles for each data column:

- function evaluation
- noise-free fitness - Fopt (and its value)
- best noise-free fitness - Fopt
- measured fitness
- best measured fitness
- x1, x2, . . . (one column for each dimension)

Fopt is the optimum of the test function considered. In the header, each of these entries are separated by the |-symbol. Each data line in the data file contains the following information:

- *1st* column - recent number of function evaluation in format %d
- *2nd* column - recent noise-free function value in format %+10.9e
- *3rd* column - best noise-free function value so far in format %+10.9e

- *4th* column - recent measured (noisy) function value in format %+10.9e
- *5th* column - best measured (noisy) function value so far in format %+10.9e
- (5+d)*th* column - value of the d*th* ($d = 1, 2, \ldots, DIM$) object parameter of the best so far noise-free function value (*3rd* column) in format %+5.4e

An example is given in Fig 6.

```
% function evaluation | noise-free fitness - Fopt (6.671000000000e+01) | best noise-free fitness - Fopt | measured fitness | best
measured fitness | x1 | x2 |...
1 +9.324567891e+05 +9.324567891e+05 +1.867342122e+06 +1.867342122e+06 +4.2345e+01 ...
2 +9.636565611e+05 +9.324567891e+05 +8.987623162e+05 +8.987623162e+05 +3.8745e+01 ...
...
31623 9.232667823e+01 9.576575761e+01 -6.624783627e+01 -1.657621581e+02 +5.1234e-02 ...
32478 1.000043784e+02 9.576575761e+01 -4.432869272e+01 -1.657621581e+02 +3.8932e-02 ...
35481 ...
...
```

Figure 6: Example of a data file

Each entry in the index files is associated to at least two data files: one for the function value-aligned data and another for the number of function evaluations-aligned data. The data file names are identical except for the file extension being '*.dat' and '*.tdat' respectively.

The writing to the function value aligned data file happens only each time the noise-free function value minus the optimum function value is less than $10^{i/5}$, for all integer i, for the first time (note, that the provided software does not return this difference to the algorithm).

The writing to the number of function evaluations aligned data file happens:

- in the first file each time the function evaluation number is equal to $\lfloor 10^{i/20} \rfloor$ for at least one $i = 1, 2, \ldots$ This means, that writing happens after about 12.2% additional function evaluations have been conducted. In particular the first 8 evaluations are written and also evaluations $\ldots$, 89, 100, 112, 125, 141,$\ldots$, 707, 794, 891, 1000, 1122,$\ldots$
- when any termination criterion is fulfilled (writing the recent evaluation and the current best so far values)

The prefix for the data file names of one experiment will be the same as the prefix of the corresponding index file. The function identifier and the dimension of the object parameters will be appended to this prefix. All data files will be saved in subfolders `data_fX`, where `X` is the function identifier, located at the same location as their index file.

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399